

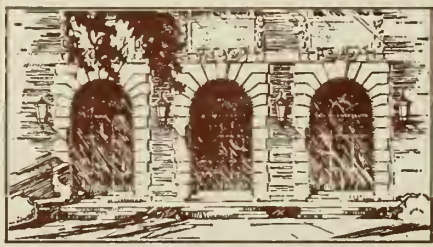
LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

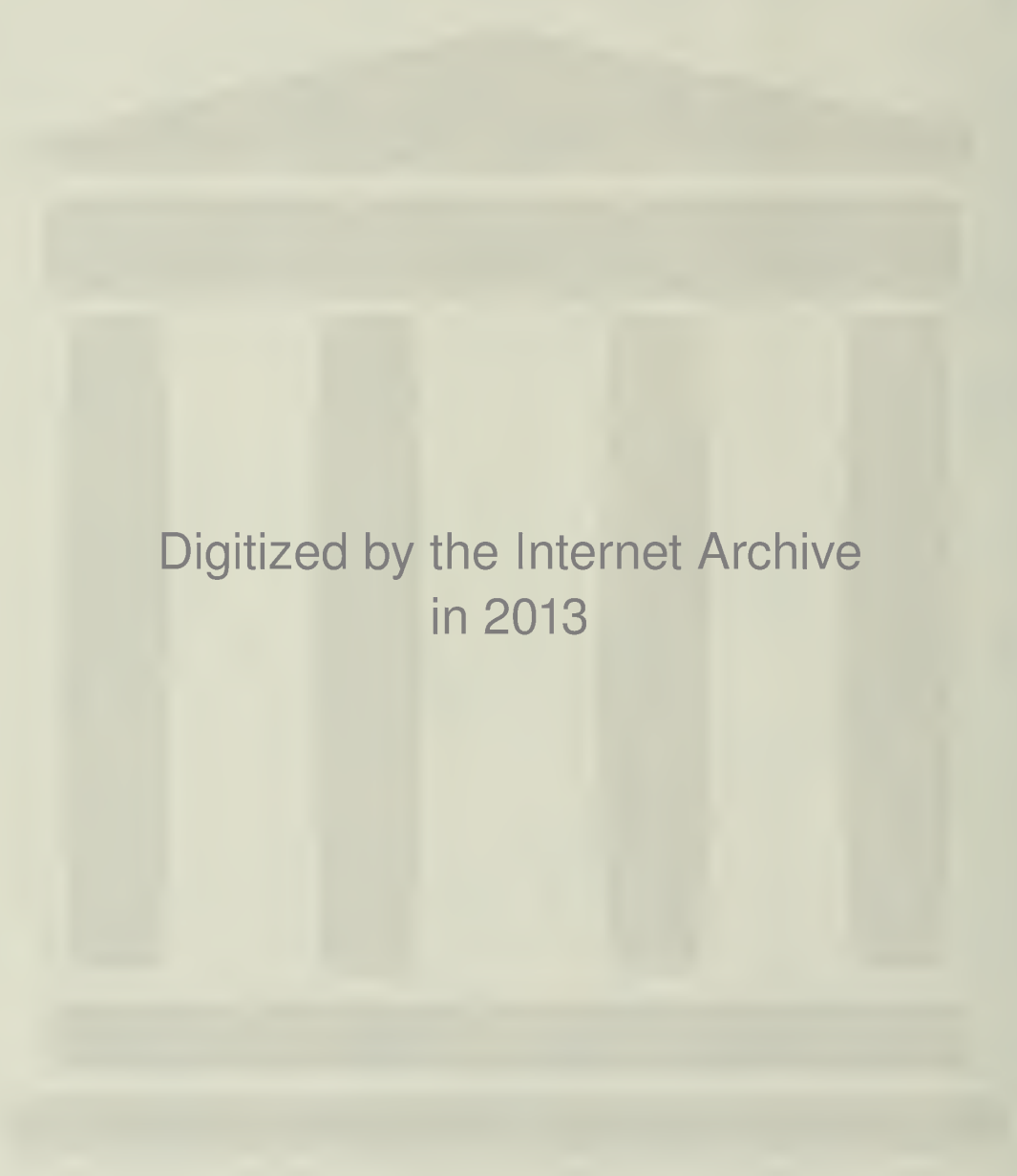
510.84

Il6r

no. 800 - 805

cop. 2





Digitized by the Internet Archive
in 2013

<http://archive.org/details/blendedlinearmul800skee>

ISGN
no. 800
COP 2

UIUCDCS-R-76-800

C00-2383-0030

BLENDING LINEAR MULTISTEP METHODS

by

Robert D. Skeel and Antony K. Kong

June 1976



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

The Library of the

AUG 11 1976

University of Illinois
at Urbana-Champaign

UIUCDCS-R-76-800

C00-2383-0030

BLENDED LINEAR MULTISTEP METHODS

by

Robert D. Skeel and Antony K. Kong

June 1976

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

This work was supported in part by the Energy Research and Development Administration under contract US ERDA AT(11-1) 2383.

ABSTRACT

The accuracy of linear multistep formulas suitable for stiff differential systems is limited. Greater accuracy can be attained by including higher derivatives in the formula, but this is not practical for all problems. However it is possible to duplicate the absolute stability region for any given m -derivative multistep formula by taking a linear combination of m multistep formulas. These blended formulas are similar to Lambert and Sigurdsson's linear multistep formulas with variable matrix coefficients, but the approach is different. Implementation details and numerical results are presented for a variable-order, variable-step blend of the Adams-Moulton and the backward differentiation formulas.

510.14
IL6r
no 300-805
cop 2

1

1. INTRODUCTION

Currently the most popular codes for solving systems of stiff ordinary differential equations are based on the backward differentiation formulas, for example Gear [9], Brayton et al [1], Hindmarsh [12], and Byrne and Hindmarsh [3]. The biggest drawback [7, p. 33] of these formulas is the poor stability properties of the higher order formulas when the Jacobian matrix for the problem has eigenvalues close to the imaginary axis. This difficulty can be overcome by restricting the order of the formulas; however these lower order formulas are often fairly inefficient because of their limited accuracy.

There are conceivably three situations in which more accurate stiffly stable formulas would be helpful:

- (i) for nonstiff problems. The casual user of a program from a subroutine library should not be required to know whether or not his problem is stiff. And there is presently no quick and reliable way for automatically determining whether or not a problem is stiff. Therefore it may be reasonable to provide a stiff formula as the default. (If efficiency were very important, the user should not be considered as "casual".)
- (ii) for stiff problems with moderate to high accuracy requirements.
- (iii) for stiff problems whose Jacobian matrix has large eigenvalues near the imaginary axis. These are the problems for which the backward differentiation formulas are often inadequate.

One can get more accurate stiffly stable formulas for a system

$$\frac{dy}{dt} = f(t, y)$$

of s differential equations by including derivatives of f in the formula:

$$\sum_{i=0}^k \alpha_i y_{n-i} = \sum_{j=1}^m h^j \sum_{i=0}^k \beta_{ji} f_{n-i}^{(j-1)}$$

where $f_{n-i}^{(j-1)} = f^{(j-1)}(t_{n-i}, y_{n-i})$. Here $f^{(\cdot)}$ denotes the total derivative of f defined recursively by

$$f^{(0)} = f, \\ f^{(\ell+1)} = \frac{\partial}{\partial t} f^{(\ell)} + f \frac{\partial}{\partial y} f^{(\ell)}.$$

Formulas of this type have been proposed by Enright [5] and Brown [2].

The presence of higher derivatives makes it necessary to require the user to provide routines for their evaluation or better still to have these routines generated symbolically from $f(t, y)$. Unfortunately there are differential equations for which symbolic differentiation is not possible, and in these cases the user may be unwilling (see [18]) or unable to provide analytic Jacobians. Thus the biggest drawback of these multiderivative formulas is their limited applicability.

Stiff problems require implicit formulas, and hence systems of nonlinear equations must be solved. This is usually accomplished by some kind of modified Newton iteration, which means that an approximation to the Jacobian is available during the integration of a stiff system. It may be advantageous to make further use of the Jacobian in our method. We cannot use it to improve the order of accuracy of the method because it may not be exact, but we can use it to enhance the *stability* properties. This idea has been pursued by Lambert [15] and by Lambert and Sigurdsson [16], who introduce the class of linear multistep formulas with variable matrix coefficients. The formulas presented in this paper are of this type, but the approach is different.

The basic idea is to take a linear multiderivative multistep formula and convert it into a variable coefficient linear multistep formula so that the region of absolute stability and the order of accuracy is unaffected. There is no reason to suspect that this transformation would either improve or degrade the performance of the formula in the case of smooth problems, although it should be beneficial for problems with nonexistent higher derivatives.

As an example, the $(k + 1)$ st order Enright formula [5] is transformed into a blend of the $(k + 1)$ st order Adams-Moulton formula and the k -th order backward differentiation formula:

$$\{\text{EnrightF}^{(k+1)}\} \rightarrow \{\text{AMF}^{(k+1)}\} + k \gamma_k^* h J\{\text{BDF}^{(k)}\}.$$

Here J is a Jacobian approximation and γ_k^* is a negative constant defined in [11, p. 195] and [10, p. 113]. By adjusting the coefficient $k \gamma_k^*$, it is possible to widen the stability region without sacrificing accuracy. Blended[†] formulas modified in this way have been put into the *CACM* algorithm DIFSUB (Gear [9]) and compared with the BDF's (see §6). The implementation of these formulas required the derivation of the Nordsieck representation (§3) and the development of local error estimators (§5).

There is a difficulty associated with the use of either multiderivative or variable matrix coefficient multistep formulas: the use of modified Newton iterations to solve the implicit equations requires "inverting" polynomials in the Jacobian matrix. This would be undesirable if sparse matrix techniques are being used, because matrix multiplications can considerably reduce sparseness. Two ways of avoiding matrix multiplications have been suggested in the literature for the case $m = 2$. Willoughby

† A term suggested by C. W. Gear

[6 , p. 102] suggests doing the LU decomposition in complex arithmetic. Enright [6] suggests choosing the parameters of the formula so that the quadratic in h J becomes a perfect square; the computation required is one LU decomposition (in real arithmetic) and *two* backsolves. A third alternative is suggested in §4, which also requires one decomposition and two backsolves. These last two methods of avoiding matrix multiplications also avoid the multiplication of a vector by a matrix, which is suggested by the appearance of $J\{BDF^{(k)}\}$ in the AMF-BDF blend.

2. DERIVATION OF BLENDED FORMULAS

We discuss two ways of converting a linear m -derivative multistep formula into a linear combination of m linear multistep formulas:

- (i) mixed order approach,
- (ii) pure order approach.

For each approach an example is presented followed by a brief discussion of the general case.

Consider Enright's third order formula:

$$y_n - y_{n-1} = \frac{2h}{3} f_n + \frac{h}{3} f_{n-1} - \frac{h^2}{6} f'_n.$$

The associated linear operator,

$$L_h y(t) = -y(t) + y(t-h) + \frac{2h}{3} y'(t) + \frac{h}{3} y'(t-h) - \frac{h^2}{6} y''(t),$$

satisfies $L_h y(t) = O(h^4)$. The main idea is to split L_h into a sum $L_h^1 + h \frac{d}{dt} L_h^2$ where L_h^1 and L_h^2 are 1-derivative operators (that is, the second derivative of $y(t)$ is not present). We choose L_h^1 so that $L_h^1 y(t) = O(h^4)$, which then forces $L_h^2 y(t) = O(h^3)$. The conditions on L_h^1 imply that

$$\begin{aligned} L_h^1 y(t) = & -y(t) + y(t-h) + h \beta_0 y'(t) + h \beta_1 y'(t-h) \\ & + \dots + h \beta_k y'(t-kh). \end{aligned}$$

where the β 's are chosen so that $L_h^1 y(t) = O(h^4)$. There is a unique choice which minimizes the stepnumber k :

$$L_h^1 y(t) = -y(t) + y(t-h) + \frac{5h}{12} y'(t) + \frac{2h}{3} y'(t-h) - \frac{h}{12} y'(t-2h),$$

which corresponds to the third order Adams-Moulton formula. This choice

of L_h^1 forces

$$L_h^2 y(t) = -\frac{1}{6} \left\{ -\frac{3}{2} y(t) + 2y(t-h) - \frac{1}{2} y(t-2h) + h y'(t) \right\}.$$

The expression in braces corresponds to the second order backward differentiation formula. Having obtained our decomposition $L_h^1 + h \frac{d}{dt} L_h^2$, we construct the blended formula by replacing L_h^1 and L_h^2 with their corresponding formulas and by replacing $\frac{d}{dt}$ with the Jacobian matrix $J = f_y(t_0, y_0)$:

$$\begin{aligned} & \left\{ -y_n + y_{n-1} + \frac{5h}{12} f_h + \frac{2h}{3} f_{n-1} - \frac{h}{12} f_{n-2} \right\} \\ & - \frac{hJ}{6} \left\{ -\frac{3}{2} y_n + 2y_{n-1} - \frac{1}{2} y_{n-2} + h f_n \right\} = 0. \end{aligned}$$

In an actual implementation J would be updated whenever f_y was reevaluated.

Two important properties of this blended formula should be noted. First, the blended formula is identical to Enright's formula for the test equation $y' = \lambda y$, and hence the region of absolute stability

$$R = \{h\lambda: y_n \rightarrow 0 \text{ as } n \rightarrow \infty \text{ for } y' = \lambda y\}$$

is the same for both formulas. Second, the normalized truncation error is

$$\left[1 - \frac{hJ}{6}\right]^{-1} \left\{ \frac{h^4}{24} y^{(4)}(\tau) - \frac{h^4}{18} J y^{(3)}(\tilde{\tau}) \right\}$$

for some τ and $\tilde{\tau}$ between $t-2h$ and t , and so the formula is of order three. Moreover, the terms in braces tend to cancel each other, particularly for the equation $y' = Jy$ for which the truncation error would be

$$\left[1 - \frac{hJ}{6}\right]^{-1} \left\{ -\frac{h^4}{72} y^{(4)}(t) + O(h^5) \right\}.$$

Here I denotes the $s \times s$ identity matrix. This convention of using scalars to represent scalar multiples of the $s \times s$ identity matrix is used throughout the paper.

Having presented an example, let us briefly look at the general case where

$$L_h y(t) = - \sum_{i=0}^k \alpha_i y(t - ih) + \sum_{j=1}^m h^j \sum_{i=0}^k \beta_{ji} y^{(j)}(t - ih) .$$

Assume the order $p \geq m - 2$, and let $q = \max\{k, p - 1\}$. We write

$$L_h = L_h^1 + h \frac{d}{dt} L_h^2 + \dots + h^{m-1} \frac{d^{m-1}}{dt^{m-1}} L_h^m$$

where

$$L_h^j y(t) = -\alpha_0^j y(t) - \dots - \alpha_q^j y(t - qh) + h \beta_0^j y'(t) + \dots + h \beta_q^j y'(t - qh) .$$

This decomposition is uniquely specified by choosing $L_h^1, L_h^2, \dots, L_h^{m-1}$ (in that order) so that

$$\beta_i^j = 0 \quad , \quad p - j < i \leq q ,$$

and

$$L_h^j y(t) = O(h^{p+2-j}) .$$

Having obtained $L_h^1, L_h^2, \dots, L_h^m$, we define the mixed order blended formula to be

$$\sum_{j=1}^m (h^j)^{j-1} \sum_{i=0}^q (-\alpha_i^j y_{n-i} + h \beta_i^j f_{n-i}) = 0 . \quad (2.1)$$

In the second approach all the formulas in the blend are of the same order. Again we illustrate by means of Enright's third order formula. The operator L_h is split into a sum $L_h^1 + h \frac{d}{dt} L_h^2$ where L_h^1 and L_h^2 are 1-derivative operators. Except this time the decomposition is uniquely specified by requiring L_h^2 to correspond to a third order formula of minimum stepnumber:

$$L_h^2 y(t) = - \frac{1}{6} \left\{ \frac{-11}{6} y(t) + 3y(t - h) - \frac{3}{2} y(t - 2h) + \frac{1}{3} y(t - 3h) + hy'(t) \right\} .$$

The expression in braces corresponds to the third order backward differentiation formula. This forces

$$L_h^1 y(t) = -y(t) + y(t-h) + \frac{13}{36} hy'(t) + \frac{5}{6} hy'(t-h) - \frac{h}{4} y'(t-2h) \\ + \frac{h}{18} y'(t-3h) ,$$

which corresponds to a linear combination of the third order Adams-Bashforth and Adams-Moulton formulas, namely $\frac{2}{15} ABF^{(3)} + \frac{13}{15} AMF^{(3)}$. The pure order blended formula is defined to be

$$\{-y_n + y_{n-1} + \frac{13}{36} hf_n + \frac{5}{6} hf_{n-1} - \frac{h}{4} f_{n-2} + \frac{h}{18} f_{n-3}\} \\ - \frac{hJ}{6} \{-\frac{11}{6} y_n + 3y_{n-1} - \frac{3}{2} y_{n-2} + \frac{1}{3} y_{n-3} + hf_n\} = 0.$$

Again this has the same absolute stability region as the third order Enright formula. Note, however, that this is a 3-step formula whereas the mixed order approach yields only a 2-step formula.

In the general case where L_h is a p -th order m -derivative k -step operator, we write

$$L_h = L_h^1 + h \frac{d}{dt} L_h^2 + \dots + h^{m-1} \frac{d^{m-1}}{dt^{m-1}} L_h^m$$

where

$$L_h^j y(t) = -\alpha_0^j y(t) - \dots - \alpha_q^j y(t-qh) + h \beta_0^j y'(t) + \dots \\ + h \beta_q^j y'(t-qh)$$

and $q = \max\{k, p\}$. The decomposition is unique if $L_n^m, L_n^{m-1}, \dots, L_n^2$ are chosen (in that order) so that

$$\alpha_i^j = 0 \quad , \quad p < i \leq q ,$$

and

$$L_h^j y(t) = O(h^{p+1}) .$$

Having obtained $L_h^1, L_h^2, \dots, L_h^m$, we define the associated pure order blended formula as in (2.1).

The Enright formulas are attractive because of their ideal stability behavior near $h\lambda \approx 0$ and near $h\lambda = \infty$, and so it would be of interest to examine the corresponding blended formulas in the general case. Enright's $(k+1)$ -st order formula

$$y_n - y_{n-1} = h \sum_{i=0}^{k-1} \beta_{1i} f_{n-i} + h^2 \beta_{20} f'_n$$

corresponds to the mixed order blended formula given by

$$\{AMF^{(k+1)}\} + k \gamma_k^* h J\{BDF^{(k)}\}$$

where

$$AMF^{(k+1)}: \quad y_n - y_{n-1} = h \sum_{i=0}^k \beta_i f_{n-i},$$

$$BDF^{(k)}: \quad \sum_{i=0}^k \alpha_i y_{n-i} = h f_n.$$

This relationship between the three formulas is easily verified once we know that $\beta_k = (-1)^k \gamma_k^*$ and $\alpha_k = (-1)^{k-1}/k$.

Note that the order of the $AMF^{(k+1)} - BDF^{(k)}$ blend is not affected by changing the coefficient $k \gamma_k^*$, and hence we consider the formula

$$\{AMF^{(k+1)}\} - \gamma^{(k)} h J\{BDF^{(k)}\}$$

where $\gamma^{(k)}$ is a free parameter. In particular $\gamma^{(k)}$ could be chosen to maximize the angle α for which the formula is $A(\alpha)$ -stable. (A formula is $A(\alpha)$ -stable if every numerical solution of $y' = \lambda y$ tends to zero for $|\text{Arg}(-\lambda)| < \alpha$.) Values for $\gamma^{(k)}$ numerically determined in this way appear in Table I in the column $\gamma^{(k)} = \gamma_{\text{opt}}^{(k)}$. The largest angle α for which each

formula is $A(\alpha)$ -stable is also given. $A(\alpha)$ -stable formulas up to order 12 were obtained. The second, third, and fourth order formulas are A -stable for a range of values of $\gamma^{(1)}$, $\gamma^{(2)}$, and $\gamma^{(3)}$. The actual choice of these three parameters is delayed until §4.

Table I. Values for $\gamma^{(k)}$ and Corresponding α -angles

Stepnumber k	Order $k + 1$	$-k \gamma_k^*$	$\gamma_{\text{opt}}^{(k)}$	α for $\gamma^{(k)} = -k \gamma_k^*$	α for $\gamma^{(k)} = \gamma_{\text{opt}}^{(k)}$
1	2	.5	$[0, +\infty)$	90°	90°
2	3	.1666667	$[\text{.1250000}, +\infty)$	90°	90°
3	4	.125	$[\text{.1218908},$ $\text{.6837917}]$	90°	90°
4	5	.1055556	.1284997	86.5°	89.4°
5	6	.09375	.1087264	79.6°	87.0°
6	7	.08561508	.09625961	72.5°	82.9°
7	8	.07957176	.08754865	61.5°	77.4°
8	9	.07485229	.08105623	40.2°	70.2°
9	10	.07103299	.07599874	not $A(0)$ -stable	60.7°
10	11	.06785850	.07192936	not $A(0)$ -stable	47.6°
11	12	.06516462	.06857227	not $A(0)$ -stable	28.7°

The second last column was obtained from measurements of Figure 3 in [4, p. 21].

The angle α is only one of a number of parameters which have been proposed for measuring the extent of the stability region R . But it is

probably the best such measure, especially for methods with automatic stepsize selection. When such methods are applied to the equation $y' = \lambda y + g(t)$ with λ complex, the integration starts out with $h\lambda$ near the origin, and as the integration proceeds, the stepsize h increases and the value $h\lambda$ moves away from the origin. However if $h\lambda$ approaches the boundary of R , this would be detected by the error estimator, and any further movement of $h\lambda$ would be prevented. The implication of this is that not all of R is "used", but only that portion which can be reached from the origin by rays lying entirely inside R . The useful part of R can be described mathematically as the set

$$\{\mu \in R: t\mu \in R \text{ for } 0 < t \leq 1\},$$

and for stiffly stable formulas this region is much like a wedge, which is best described by the angle α . This parameter also serves as a good indicator of the problem class for which the formula is suitable, namely those problems for which the "large" eigenvalues λ of the Jacobian satisfy $|\text{Arg}(-\lambda)| < \alpha$. Ideally we would like to have $\alpha = \frac{\pi}{2}$ so that the formula would be suitable for all realistic problems.

It may be helpful to present the original motivation for the $\text{AMF}^{(k+1)} - \text{BDF}^{(k)}$ blend. We begin by considering a scalar equation $y' = f(t, y)$. If the stepsize h were determined only by accuracy considerations, then the size of $-h f_y$ would be a good measure of stiffness. If $-h f_y$ were small, then the $\text{AMF}^{(k+1)}$ would be a good k -step formula to use, and if $-h f_y$ were large then the $\text{BDF}^{(k)}$ would be a good choice. And so it would seem that a weighted sum

$$\{\text{AMF}^{(k+1)}\} + \gamma^{(k)}(-h f_y)\{\text{BDF}^{(k)}\}$$

would give us the best of both formulas. For small $-h f_y$ this blend is nearly the $AMF^{(k+1)}$, and for large $-h f_y$ it is nearly the $BDF^{(k)}$.

We now extend this idea to a system. Consider a change of coordinates which decouples the system at $(t, y) = (t_n, y_n)$. Such a coordinate transformation can be accomplished by a matrix T which diagonalizes $f_y(t_n, y_n)$; that is, $T^{-1} f_y(t_n, y_n) T = \Lambda$ where Λ is diagonal. This is almost always possible. Letting $z = T^{-1} y$, the system becomes

$$z' = g(t, z) = T^{-1} f(t, Tz).$$

Since this system is locally decoupled, we can use $-h \Lambda^{ii}$ as a measure of the stiffness of the i -th differential equation. Each equation of the system can be integrated by a formula which is suitable for that equation:

$$(-z_n^i + z_{n-1}^i + h \sum_{j=0}^k \beta_j g_{n-j}^i) + \gamma^{(k)} (-h \Lambda^{ii}) (-\sum_{j=0}^k \alpha_j z_{n-j}^i + h g_n^i) = 0.$$

Transforming this back to the original coordinate system gives

$$(-y_n + y_{n-1} + h \sum_{j=0}^k \beta_j f_{n-j}) + \gamma^{(k)} (-h f_y) (-\sum_{j=0}^k \alpha_j y_{n-j} + h f_n) = 0,$$

which is the blended formula.

The remainder of the paper is restricted to a discussion of the $AMF^{(k+1)}$ - $BDF^{(k)}$ blend with $\gamma^{(k)} = \gamma_{opt}^{(k)}$.

3. THE NORDSIECK REPRESENTATION

In the Nordsieck representation we work with an approximation

$\underline{y}_n = [y_n, h y'_n, \dots, h^k y_n^{(k)}/k!]^T$ to the scaled derivatives $[y(t_n), h y'(t_n), \dots, h^k y^{(k)}(t_n)/k!]^T$ of the solution. A linear Nordsieck formula can be thought of as consisting of a prediction

$$\underline{p}_n = P \underline{y}_{n-1}$$

followed by a correction

$$\underline{y}_n = \underline{p}_n + \underline{\ell} \Delta_n$$

where Δ_n is chosen so that $y'_n = f(t_n, y_n)$; that is, Δ_n is defined implicitly by

$$\Delta_n = \ell_1^{-1} \{h f(t_n, p_n + \ell_0 \Delta_n) - h p'_n\}.$$

Here $\underline{p}_n = [p_n, h p'_n, \dots, h^k p_n^{(k)}/k!]^T$, $\underline{\ell} = [\ell_0, \ell_1, \dots, \ell_k]^T$, and P is the Pascal triangle matrix given by

$$P_{ij} = \binom{j}{i}, \quad 0 \leq i, j \leq k.$$

Nordsieck formulas are closely related to multistep formulas. For every linear k -step formula of order $\geq k$ there is a unique $(k+1)$ -value linear Nordsieck formula such that the values $y_n, y_{n-1}, \dots, y_{n-k}$ generated by the Nordsieck formula from $\underline{y}_{n-k}$ satisfy the difference equation of the multistep formula. Let the multistep formula be

$$\rho(E) y_{n-k} = \sigma(E) h f_{n-k}$$

where E is the shift operator,

$$\rho(\xi) = \sum_{i=0}^k \alpha_i \xi^{k-i}, \quad \text{and} \quad \sigma(\xi) = \sum_{i=0}^k \beta_i \xi^{k-i}.$$

The corrector vector $\underline{\ell}$ of the "equivalent" Nordsieck formula satisfies

$$\rho(\xi) = (\xi - 1)^{k+1} \underline{e}_1^T (\xi I - P)^{-1} \underline{x} \quad (3.1)$$

and

$$\sigma(\xi) = (\xi - 1)^{k+1} \underline{e}_0^T (\xi I - P)^{-1} \underline{x} \quad (3.2)$$

where $\underline{e}_1^T = [0, 1, 0, \dots, 0]$ and $\underline{e}_0^T = [1, 0, 0, \dots, 0]$. It can be shown from these equations that

$$\lambda_1 = \alpha_0 \quad \text{and} \quad \lambda_0 = \beta_0. \quad (3.3)$$

Proofs of these relationships will appear in [19].

Let $\underline{a}^{(k)}$ and $\underline{b}^{(k)}$ denote the corrector vector corresponding to the k -step AMF^(k+1) and BDF^(k) respectively. It can be shown [19] that

$$a_j^{(k)} = \frac{1}{j(k-1)!} \left[\begin{matrix} k \\ j \end{matrix} \right], \quad j = 1(1)k,$$

$$a_0^{(k)} = - \sum_{j=1}^{k+1} (-1)^j a_j^{(k+1)}$$

and

$$b_j^{(k)} = \frac{1}{k!} \left[\begin{matrix} k+1 \\ j+1 \end{matrix} \right], \quad j = 0(1)k,$$

where the brackets denote Stirling numbers of the first kind (see e.g. Knuth [13, p. 66]). Values of $\underline{a}^{(k)}$ and $\underline{b}^{(k)}$ are given in Appendix A for $k = 1(1)11$.

Consider the Nordsieck formula with $\underline{x} = \underline{a} - \gamma h J \underline{b}$, where we have suppressed the superscripts (k) . Applying (3.1) and (3.2) gives $\rho(\xi) = \rho^{AM}(\xi) - \gamma h J \rho^{BD}(\xi)$ and $\sigma(\xi) = \sigma^{AM}(\xi) - \gamma h J \sigma^{BD}(\xi)$. Therefore the values computed by this Nordsieck formula satisfy

$$\rho^{AM}(E) y_{n-k} - h \sigma^{AM}(E) f_{n-k} - \gamma h J \{ \rho^{BD}(E) y_{n-k} - h \sigma^{BD}(E) f_{n-k} \} = 0,$$

which is the difference equation for an AMF^(k+1) - BDF^(k) blend.

In an actual implementation J might be reevaluated every few steps; in other words,

$$y_n = P y_{n-1} + \underline{\ell}_n \Delta_n \quad (3.4)$$

where $\underline{\ell}_n = \underline{a} - \gamma h J_n \underline{b}$. And a blended Nordsieck formula with a changing J is *not* equivalent to a blended multistep formula with a changing J . For example, the blended 3-value Nordsieck formula generates values y_n, y_{n-1}, y_{n-2} which satisfy

$$\begin{aligned} & (1 + 3\gamma h J_n)^{-1} [y_n - y_{n-1} - \frac{5h}{12} f_n - \frac{7h}{12} f_{n-1} \\ & - \gamma h J_n \{ \frac{3}{2} y_n - \frac{3}{2} y_{n-1} - h f_n - \frac{h}{2} f_{n-1} \}] \\ & + (1 + 3\gamma h J_{n-1})^{-1} [\frac{-h}{12} f_{n-1} + \frac{h}{12} f_{n-2} \\ & - \gamma h J_{n-1} \{ -\frac{1}{2} y_{n-1} + \frac{1}{2} y_{n-2} + \frac{h}{2} f_{n-1} \}] = 0. \end{aligned} \quad (3.5)$$

This difference equation can be derived as follows. Premultiply (3.4) by $\underline{e}_1^T$ to get

$$h y'_n = \underline{e}_1^T P y_{n-1} + \underline{\ell}_{1n} \Delta_n,$$

and substitute this back into (3.4):

$$y_n = (I - \underline{\tilde{\ell}}_n \underline{e}_1^T) P y_{n-1} + \underline{\tilde{\ell}}_n h y'_n$$

where $\underline{\tilde{\ell}}_n = \underline{\ell}_{1n}^{-1} \underline{\ell}_n$. From this equation it follows that

$$y_{n-1} = \underline{e}_0^T (I - \underline{\tilde{\ell}}_{n-1} \underline{e}_1^T) P y_{n-2} + \underline{e}_0^T \underline{\tilde{\ell}}_{n-1} h y'_{n-1}$$

and

$$\begin{aligned} y_n &= \underline{e}_0^T (I - \underline{\tilde{\ell}}_n \underline{e}_1^T) P (I - \underline{\tilde{\ell}}_{n-1} \underline{e}_1^T) P y_{n-2} \\ &+ \underline{e}_0^T (I - \underline{\tilde{\ell}}_n \underline{e}_1^T) P \underline{\tilde{\ell}}_{n-1} h y'_{n-1} + \underline{e}_0^T \underline{\tilde{\ell}}_n h y'_n, \end{aligned}$$

which is a system of two equations relating y_n , y_{n-1} , y_{n-2} , $h y'_n$, $h y'_{n-1}$, $h y'_{n-2}$, and $h^2 y''_{n-2}/2$. Eliminating $h^2 y''_{n-2}/2$ and substituting f_n for y'_n yields equation (3.5).

4. THE CORRECTOR ITERATION

The implicit equation satisfied by Δ_n is

$$h p' + \ell_1 \Delta - h f(t, p + \ell_0 \Delta) = 0$$

where for convenience we suppress dependence on n . The Newton iteration for this equation is

$$\Delta_{(0)} = 0,$$

$$\Delta_{(m+1)} = \Delta_{(m)} + [\ell_1 - h(f_y)_{(m)} \ell_0]^{-1} \{h f_{(m)} - (h p' + \ell_1 \Delta_{(m)})\}$$

where for a function the subscript (m) denotes evaluation at $(t, p + \ell_0 \Delta_{(m)})$.

This iteration is the same for both the multistep form and the Nordsieck form of the formula. Convergence of this iteration is seldom a problem, and so a considerable saving in time results if a recent Jacobian approximation J is used in place of $(f_y)_{(m)}$:

$$\Delta_{(m+1)} = \Delta_{(m)} + [\ell_1 - h J \ell_0]^{-1} \{h f_{(m)} - (h p' + \ell_1 \Delta_{(m)})\}. \quad (4.1)$$

From $\underline{\ell} = \underline{a} - \gamma h J \underline{b}$ and from (3.3) we have that

$$\ell_1 - h J \ell_0 = (1 - \gamma \alpha h J) - h J(\beta - \gamma h J)$$

where $\alpha = \alpha_0^{BD}$ and $\beta = \beta_0^{AM}$. This means that (4.1) involves the LU decomposition of a quadratic polynomial in $h J$. But forming matrix products can be expensive, and this is particularly true if sparse matrix techniques are being used because matrix multiplications reduce sparseness.

One way of avoiding matrix multiplications is suggested by Willoughby [6, p. 102], and it is applicable as long as the roots of $1 - (\gamma \alpha + \beta)z + \gamma z^2$ are complex conjugates. This technique calls for an LU decomposition in complex arithmetic, which has the disadvantage of being four times slower and requiring twice as much storage compared to real arithmetic.

A second possibility is to choose the parameter γ so that $1 - (\gamma \alpha + \beta)z + \gamma z^2$ is a perfect square $(1 - cz)^2$. Then multiplication by $(1 - chJ)^{-2}$ can be accomplished by decomposing $1 - chJ$ and performing two backsolves. Enright [6] derives a second set of formulas based on this idea. This approach, however, reduces the number of free parameters, and so one order of accuracy must be sacrificed.

There is a third way of avoiding matrix multiplications, and that is to approximate $1 - (\gamma \alpha + \beta)z + \gamma z^2$ by a perfect square $(1 - cz)^2$ so that instead of (4.1) we use

$$\Delta_{(m+1)} = \Delta_{(m)} + [1 - chJ]^{-2} \{h f_{(m)} - (h p' + \ell_1 \Delta_{(m)})\}. \quad (4.2)$$

This is the iteration which is used in our implementation of the blended formulas.

Recall that for $k = 1, 2, 3$ there is some freedom in the choice of γ . We use this freedom to make $1 - (\gamma \alpha + \beta)z + \gamma z^2$ as close as possible to a perfect square. There are two choices of γ which yield a perfect square:

$$\gamma^{(1)} = \frac{3}{2} \pm \sqrt{2} \quad ,$$

$$\gamma^{(2)} = \frac{11}{18} \pm \frac{2}{9} \sqrt{6} \quad ,$$

$$\gamma^{(3)} = \frac{189}{484} \pm \frac{18}{121} \sqrt{5} \quad .$$

The smaller value of γ in each case results in a smaller truncation error, and so we choose the γ 's as close as possible to the smaller value subject to the constraints imposed by A-stability (see Table I):

$$\gamma^{(1)} = \frac{3}{2} - \sqrt{2} = .08578644 \quad ,$$

$$\gamma^{(2)} = .1250000 \quad ,$$

$$\gamma^{(3)} = .1218908 \quad .$$

The iteration (4.2) is significantly different from (4.1) in that it converges linearly for the system $y' = J y$ whereas (4.1) converges after one iteration for this system. Thus it is important to choose c so that convergence is rapid for this special case. In practice the eigenvalues of $h J$ have either a small modulus or a negative real part. However, it is only those eigenvalues with negative real part which might cause convergence difficulties. Hence the value of c is chosen by considering the test equation $y' = \lambda y$, $\text{Re } \lambda \leq 0$. For this problem the iteration given by (4.2) becomes

$$\Delta_{(m+1)} = \Delta_{(m)} + [1 - c h \lambda]^{-2} \{ h \lambda [p + (\beta - \gamma h \lambda) \Delta_{(m)}] - [h p' + (1 - \gamma h \lambda \alpha) \Delta_{(m)}] \}.$$

Letting $\mu = h \lambda$ and simplifying gives

$$\Delta_{(m+1)} = [1 - c \mu]^{-2} \{ (\gamma \alpha + \beta - 2c) \mu + (c^2 - \gamma) \mu^2 \} \Delta_{(m)} + [1 - c \mu]^{-2} h(\lambda p - p').$$

The rate of convergence depends on the coefficient of $\Delta_{(m)}$, and so we choose c to minimize

$$\phi(c) = \max_{\text{Re } \mu \leq 0} |[1 - c \mu]^{-2} \{ (\gamma \alpha + \beta - 2c) \mu + (c^2 - \gamma) \mu^2 \}|.$$

Clearly we must choose $c > 0$, and so the expression inside the vertical bars is analytic for $\text{Re } \mu \leq 0$. By the maximum modulus theorem the maximum is attained for $\text{Re } \mu = 0$, and so

$$\phi(c) = \max_{-\infty < \omega < +\infty} \frac{|(\gamma \alpha + \beta - 2c) i \omega - (c^2 - \gamma) \omega^2|}{|1 - c i \omega|^2}.$$

Equivalently

$$[\phi(c)]^2 = \max_{\omega} \frac{\omega^2 [(\gamma \alpha + \beta - 2c)^2 + (c^2 - \gamma)^2 \omega^2]}{[1 + c^2 \omega^2]^2}.$$

Differentiation with respect to ω yields extrema at

$$\omega^2 = 0,$$

$$\omega^2 = \infty,$$

and

$$\omega^2 = \frac{(\gamma \alpha + \beta - 2c)^2}{c^2(\gamma \alpha + \beta - 2c)^2 - 2(c^2 - \gamma)^2} \quad \text{if } 2(c^2 - \gamma)^2 < c^2(\gamma \alpha + \beta - 2c)^2;$$

therefore

$$[\phi(c)]^2 = \begin{cases} \frac{(c^2 - \gamma)^2}{c^4} & \text{if } 2(c^2 - \gamma)^2 \geq c^2(\gamma \alpha + \beta - 2c)^2, \\ \max \left\{ \frac{(c^2 - \gamma)^2}{c^4}, \frac{(\gamma \alpha + \beta - 2c)^4}{4[c^2(\gamma \alpha + \beta - 2c)^2 - (c^2 - \gamma)^2]} \right\} & \text{otherwise.} \end{cases}$$

The value of c which minimizes $\phi(c)$ was determined numerically for $k = 1(1)11$. These values appear in Table II.

Table II. Values for $c^{(k)}$.

Stepnumber k	$c^{(k)}$	$\phi(c^{(k)})$
1	.2928932	0
2	.3374973	.1184662
3	.3335427	.1161818
4	.3427329	.1139836
5	.3169058	.0995555
6	.2992971	.0894459
7	.2862392	.0819169
8	.2760327	.0760633
9	.2677630	.0713660
10	.2608834	.0675036
11	.2550426	.0642651

From Table II we see that even in the worst case, λ pure imaginary and $k = 2$, the asymptotic error coefficient is only .118, which means that each iteration is good for an additional 0.93 digits.

5. ERROR ESTIMATION AND CONTROL

The purpose of estimating the local error is to assist in the selection of stepsize and order in such a way that

- (i) the least possible work is done for the accuracy attained.
- (ii) the global error behaves in a reasonable manner as a function of the local error tolerance.

Hence the estimation of the local error is not in itself of much importance.

The theory of error estimation is not well developed. Usually an attempt is made to devise a local error estimate which is asymptotically equal to the local error under the assumption that the order and stepsize do not vary. This approach must be used cautiously. For example the (strongly A-stable) formula $y_n = y_{n-1} + .501 h f_{n+1} + .499 h f_n$ has the asymptotically correct error estimate $.001 h(f_{n+1} - f_n)$. However, this will often give a very poor estimate, simply because the leading term in the asymptotic expansion of the local error is not the dominant source of error unless h is very small. A more realistic approach would be to make use of a good *bound* on the truncation error. This approach would yield the estimate $.250001 h ||f_n - f_{n-1}||$ for our example. It is worth noting that the highly rated code DE of Shampine and Gordon [17] uses error estimates that are not asymptotically correct; for example, the estimate $.5 h(f_n - f_{n-1})$ is used for the trapezoid formula.

Our implementation of the blended formulas uses an error bound as the basis for the error estimate. In order to derive the error bound, we first split the $AMF^{(k+1)}$ into a sum of the $AMF^{(k)}$ and the $ABF^{(k)}$:

$$AMF^{(k+1)} = \frac{\gamma_k}{\gamma_{k-1}} AMF^{(k)} + \frac{\gamma_{k-1} - \gamma_k}{\gamma_{k-1}} ABF^{(k)}$$

where γ_k is given in Henrici [11, p. 193] and Gear [10, p. 108]. This equality follows from the fact that $\beta_0 = \gamma_k$ for the AMF^(k+1). From Henrici [11, equation (5-22)] we have that $\gamma_k - \gamma_{k-1} = \gamma_k^*$, and hence

$$\text{AMF}^{(k+1)} = \frac{\gamma_k}{\gamma_{k-1}} \text{AMF}^{(k)} - \frac{\gamma_k^*}{\gamma_{k-1}} \text{ABF}^{(k)} .$$

If we consider the AMF-BDF blend applied to a scalar equation, we get that the (normalized) truncation error is

$$h^{k+1} (1 - \gamma h \lambda)^{-1} \left[\frac{\gamma_k \gamma_k^*}{\gamma_{k-1}} y^{(k+1)}(\tau) - \frac{\gamma_k \gamma_k^*}{\gamma_{k-1}} y^{(k+1)}(\tau') - \frac{\gamma h \lambda}{k+1} y^{(k+1)}(\tau'') \right] .$$

And since

$$2 \left| \frac{\gamma_k \gamma_k^*}{\gamma_{k-1}} \right| \leq \frac{1}{k+1} \quad \text{for} \quad k = 1(1)11 ,$$

we have the bound

$$\frac{h^{k+1}}{k+1} t_{n-k} \leq t \leq t_n \quad |y^{(k+1)}(t)|$$

on the truncation error when λ is real and negative. Hence we use

$$\frac{h_n^k}{k+1} ||y_n^{(k)} - y_{n-1}^{(k)}||$$

as our error estimate.

6. COMPARISON WITH THE BACKWARD DIFFERENTIATION FORMULAS

We begin by examining the changes that must be made to a BDF algorithm in order to convert it into an algorithm for the AMF-BDF blend. The prediction step is identical for the two formulas, and it is only in the correction step that there are differences. The BDF's have a correction step something like

```

 $\Delta \leftarrow 0$ 
for  $m \leftarrow 1, 2, 3$ 
    {
         $f \leftarrow f(t, y)$ 
        if desirable then {
             $W \leftarrow b_1 - h \times f_y(t, y)$ 
            decompose  $W$ 
            solve  $W \times d = h \times f - hy'$  for  $d$ 
        }
        do {
             $y \leftarrow y + d$ 
             $hy' \leftarrow hy' + b_1 \times d$ 
             $\Delta \leftarrow \Delta + d$ 
            if  $d$  small enough then go to testerror
        }
        deal with iteration convergence failure
    }
testerror: if  $\Delta/(k+1)$  too large then go to tryagain
           for  $j \leftarrow 2(1)k$  do  $h^j y^{(j)}/j! \leftarrow h^j y^{(j)}/j! + b_j \times \Delta$ 

```

whereas for the AMF-BDF blend we have

$\Delta \leftarrow 0; \tilde{\Delta} \leftarrow 0$

for $m \leftarrow 1, 2, 3$

$f \leftarrow f(t, y)$
if desirable then $\left\{ \begin{array}{l} cH \leftarrow c \times h \\ W \leftarrow 1 - cH \times f_y(t, y) \\ \text{decompose } W \end{array} \right.$
 solve $W \times \tilde{d} = h \times f - hy'$ for $\tilde{d}$
do $\left\{ \begin{array}{l} \text{solve } W \times d = \tilde{d} \text{ for } d \\ \tilde{d} \leftarrow \gamma \times h \times (\tilde{d} - d)/cH \\ y \leftarrow y + a_0 \times d + \tilde{d} \\ hy' \leftarrow hy' + d + b_1 \times \tilde{d} \\ \Delta \leftarrow \Delta + d; \tilde{\Delta} \leftarrow \tilde{\Delta} + \tilde{d} \\ \text{if } d + \tilde{d} \text{ small enough then go to testerror} \end{array} \right.$

deal with iteration convergence failure

testerror: if $(\Delta + \tilde{\Delta})/(k + 1)$ too large then go to tryagain

for $j \leftarrow 2(1)k$ do $h^j y^{(j)}/j! \leftarrow h^j y^{(j)}/j! + a_j \times \Delta + b_j \times \tilde{\Delta}$

In both cases one function evaluation is required for each iteration.

However the corrector overhead for the AMF-BDF blend is twice as great as for the BDF. The overhead is defined to be any computation other than function evaluations.

A significant portion of the overhead of a stiff system solver often consists of LU decompositions. We would not expect there to be a difference between the two formulas in the number of decompositions if we were to use the same type of iteration in both cases. For the BDF we use

$$\Delta_{(m+1)} = \Delta_{(m)} + [x_1 - H J x_0]^{-1} \{ \dots \}$$

where H is the size of the step when J was last evaluated. The same type of iteration for the blended formula would be

$$\Delta_{(m+1)} = \Delta_{(m)} + [(1 - \gamma \alpha h J) - H J(\beta - \gamma h J)]^{-1} \{ \dots \}$$

However what we actually use is

$$\Delta_{(m+1)} = \Delta_{(m)} + [1 - c H J]^{-2} \{ \dots \}.$$

This second iteration might be expected to fail more often than the first, but it is difficult to analytically compare the two iterations even for the equation $y' = \lambda y$, and therefore we have to rely mainly on empirical evidence.

If the preceding discussion is generalized to blends of m formulas, it can be seen that the correction iteration overhead is proportional to m . Hence blends of m formulas are competitive only if they require significantly fewer function evaluations to achieve a given accuracy. How many fewer depends on the ratio of the cost of a function evaluation to the cost of a backsolve. For problems with very simple right hand sides $f(t, y)$, the cost of a function evaluation might be roughly equal to the cost of a backsolve and in this case it would not pay to use a blend of m formulas unless the number of function evaluations compared to that of the BDF's is smaller by a factor of $2/(m + 1)$. Here we are assuming that the number of decompositions is independent of m , which is fairly reasonable since it is primarily the heuristics used by the program that determine the number of decompositions.

We have compared the efficiency of the blended formulas to that of the BDF's on several test problems. In order to make the comparison fair, the blended formulas were implanted into a well-known BDF code (Gear's DIFSUB [9]) in such a way that only the formula-dependent part of the code

was changed. And therefore the code was not tuned for the blended formulas. There was one modification made to the codes and that was to replace the call to MATINV by calls to DECOMP and SOLVE [8]. A listing of the source code appears in Appendix B.

In order to facilitate testing, only problems with known analytic solutions were used. The stepsize h was initially set to the local error tolerance ϵ , and on subsequent steps it was set to $\min\{h, t_f - t\}$ where h is the stepsize recommended by the method and $[0, t_f]$ is the interval of integration. The parameter $h_{\max}$ was set to $t_f - t$ and $h_{\min}$ was (mistakenly) set to $\frac{1}{4} u t_f$ where u is the unit roundoff error, 16^{-13} , of the IBM 360. The weight vector w was initialized to all ones. The Jacobian $f_y(t, y)$ was evaluated numerically. The relative (global) error was defined by

$$\max_{0 \leq n \leq N} \left(\sum_{i=1}^s \left(\frac{y_n^i - y^i(t_n)}{w_n^i} \right)^2 \right)^{1/2}$$

where $w_n^i = \max\{1, y_0^i, y_1^i, \dots, y_n^i\}$.

Tables III through VII give the results of comparing DIFSUB with blended DIFSUB for five different types of problems. Each problem was integrated for nine different error tolerances, $\epsilon = 10^{-2}, 10^{-3}, \dots, 10^{-10}$. For each integration the following statistics were collected:

max order	- the order of the highest order formula selected by the code.
steps	- the total number of steps taken.
func evals	- the total number of function calls (including those needed to approximate the Jacobian).
back solves	- the total number of calls to the subroutine SOLVE, which solves a linear system that is in factored form.
LU decomp	- the total number of calls to the subroutine DECOMP, which performs an LU factorization on a matrix.

accurate digits - the negative of the base 10 logarithm of the relative (global) error, which was defined previously.

time - machine time in centiseconds on the IBM 360 model 75 at the University of Illinois at Urbana-Champaign. These timings are not completely reliable due to multiprogramming.

It should be stressed that the intent is to compare two classes of *formulas* and not two methods or two programs. Also, only one aspect of the relative utility of the two formulas is being compared, namely machine time versus global accuracy.

Problem I. The following is a linear problem whose Jacobian matrix has eigenvalues $-.1, -50, -120$:

$$y' \begin{bmatrix} -.1 & -49.9 & 0 \\ 0 & -50 & 0 \\ 0 & 70 & -120 \end{bmatrix} y \quad y(0) = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}$$

This was integrated on $[0, 15]$.

Table III. Numerical Results for Problem 1

ϵ	max order	steps	func evals	back solves	LU decomps	accurate digits	time
DIFSUB							
10^{-2}	3	29	104	76	9	1.9	30
10^{-3}	5	49	145	108	12	2.9	38
10^{-4}	4	70	202	171	10	3.9	51
10^{-5}	6	106	286	240	15	5.0	75
10^{-6}	6	193	508	453	18	5.7	128
10^{-7}	6	176	474	422	17	6.7	128
10^{-8}	6	204	551	505	15	7.5	143
10^{-9}	6	270	771	719	17	8.4	185
10^{-10}	6	353	1024	969	18	9.3	251
blended DIFSUB							
10^{-2}	4	30	101	146	9	3.2	33
10^{-3}	5	43	141	220	10	3.9	45
10^{-4}	6	69	195	316	12	4.6	71
10^{-5}	7	89	238	396	13	5.4	86
10^{-6}	8	120	308	524	15	6.5	110
10^{-7}	11	139	410	662	26	7.4	155
10^{-8}	9	169	443	782	17	8.5	165
10^{-9}	11	209	540	952	21	9.3	228
10^{-10}	11	234	595	1056	22	10.4	261

An examination of the last two columns reveals that there is a slight though definite improvement in performance when the blended

formulas are substituted for the BDF's. The reason that blended DIFSUB does more LU decompositions might be due to the fact that fresh decompositions are performed whenever the order is changed but not when the stepsize is changed.

Problem II. The following is problem 12 in Krogh's [14] set of test problems; it is also used by Gear [10, p. 218]:

$$y' = U^T \begin{bmatrix} -1000z^1 + (z^1)^2 \\ -800z^2 + (z^2)^2 \\ 10z^3 + (z^3)^2 \\ -.001z^4 + (z^4)^2 \end{bmatrix} \quad y(0) = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

where $z = Uy$ and

$$U = \frac{1}{2} \begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix}$$

This was integrated over $[0, 1000]$. The eigenvalues of the Jacobian matrix are equal to -1002 , -802 , 8 , and -2.001 at $t = 0$, and approach -1000 , -800 , -10 , and $-.001$ respectively as $t \rightarrow \infty$.

Table IV. Numerical Results for Problem 2

ϵ	max order	steps	func evals	back solves	LU decomps	accurate digits	time
DIFSUB							
10^{-2}	3	64	277	184	23	1.7	65
10^{-3}	4	105	374	273	25	2.7	91
10^{-4}	5	167	496	387	27	3.6	131
10^{-5}	6	250	733	608	31	4.1	206
10^{-6}	6	284	778	677	25	5.4	229
10^{-7}	6	380	1076	915	40	6.4	318
10^{-8}	6	467	1311	1178	33	7.0	416
10^{-9}	6	590	1641	1516	31	8.2	586
10^{-10}	6	768	2083	1954	32	9.1	699
blended DIFSUB							
10^{-2}	4	61	276	358	24	2.7	75
10^{-3}	6	98	333	488	22	3.7	105
10^{-4}	7	146	449	688	26	4.6	160
10^{-5}	7	200	594	970	27	5.4	231
10^{-6}	9	262	750	1242	32	6.4	314
10^{-7}	9	325	900	1502	37	7.3	419
10^{-8}	9	381	1062	1818	38	8.4	521
10^{-9}	11	474	1303	2244	45	9.4	734
10^{-10}	11	558	1548	2678	52	10.4	852

Once again there is a slight though definite improvement when the blended formulas are substituted for the BDF's.

Problem III. This is problem B5 in the set of stiff problems considered by Enright et al [7]:

$$y' = \begin{bmatrix} -10 & 100 & 0 & 0 & 0 & 0 \\ -100 & -10 & 0 & 0 & 0 & 0 \\ 0 & 0 & -4 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & -.1 \end{bmatrix} y \quad y(0) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

This was integrated over the interval $[0, 20]$. Because the Jacobian matrix has eigenvalues $-10 \pm i100$, -4 , -1 , $-.5$, $-.1$, this problem can be troublesome for the BDF's.

Table V. Numerical Results for Problem 3

ϵ	max order	steps	func evals	back solves	LU decomps	accurate digits	time
DIFSUB							
10^{-2}	(4)	(1001)	(3002)	(2959)	(7)	(1.1)	
"too much work" ; integration stopped at $t = 8.3$							
blended DIFSUB							
10^{-2}	6	125	493	756	19	2.9	215
10^{-3}	6	194	691	1152	19	3.7	348
10^{-4}	7	277	922	1590	21	4.5	517
10^{-5}	8	368	1196	2114	23	5.4	729
10^{-6}	9	468	1494	2686	25	6.4	1040
10^{-7}	11	592	1831	3288	31	7.3	1546
10^{-8}	11	721	2178	3958	33	8.5	1772
10^{-9}	12	895	2644	4878	34	9.4	2369
10^{-10}	(12)	(1001)	(2917)	(5580)	(21)	(10.3)	
"too much work" ; integration stopped at $t = 5.1$							

There is a remarkable difference between the abilities of the two types of formulas to solve this problem.

Problem IV. The following is problem 13 in Krogh's set of problems; it is also used by Enright et al [7, problem E4]:

$$y' = U^T \begin{bmatrix} 10z^1 + 10z^2 + \frac{1}{2}(z^1)^2 - \frac{1}{2}(z^2)^2 \\ -10z^1 + 10z^2 + z^1 z^2 \\ -1000z^3 + (z^3)^2 \\ -.01z^4 + (z^4)^2 \end{bmatrix} \quad y(0) = \begin{bmatrix} 0 \\ -2 \\ -1 \\ -1 \end{bmatrix}$$

where $z = Uy$ and

$$U = \frac{1}{2} \begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix}$$

This was integrated over $[0,1000]$. The eigenvalues of the Jacobian matrix are equal to $8 \pm 10i$, -1002 , and -2.01 at $t = 0$ and approach $-10 \pm 10i$, -100 , and $-.01$ as $t \rightarrow \infty$.

Table VI. Numerical Results for Problem 4

ϵ	max order	steps	func evals	back solves	LU decomps	accurate digits	time
DIFSUB							
10^{-2}	3	79	388	263	31	1.3	78
10^{-3}	4	137	555	402	38	2.0	116
10^{-4}	6	289	836	715	30	3.3	249
10^{-5}	6	268	802	677	31	4.2	229
10^{-6}	(6)	(1001)	(2777)	(2652)	(31)	(5.0)	
"too much work" ; integration stopped at $t = 150$							
blended DIFSUB							
10^{-2}	4	79	347	484	26	2.6	95
10^{-3}	5	127	463	708	27	3.5	140
10^{-4}	7	170	558	890	28	4.7	191
10^{-5}	7	244	750	1258	30	5.4	266
10^{-6}	7	325	952	1646	32	6.4	376
10^{-7}	9	399	1165	2008	40	7.4	536
10^{-8}	11	483	1367	2364	46	8.3	724
10^{-9}	(11)	(1001)	(5511)	(6708)	(539)	(9.2)	1389
"too much work" ; integration stopped at $t = 977$							

For this problem the blended formulas performed significantly better.

Problem V. The following problem (problem 8 of Krogh's set of test problems) tests the integrator's ability to solve nonstiff problems:

$$y' = \begin{bmatrix} y^3 \\ y^4 \\ -y^1((y^1)^2 + (y^2)^2)^{-3/2} \\ -y^2((y^1)^2 + (y^2)^2)^{-3/2} \end{bmatrix} \quad y(0) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

This was integrated over $[0, 20]$.

Table VII. Numerical Results for Problem 5

ϵ	max order	steps	func evals	back solves	LU decomps	accurate digits	time
DIFSUB							
10^{-2}	4	50	323	210	28	-0.4	73
10^{-3}	5	81	345	264	20	0.4	91
10^{-4}	5	95	453	324	32	0.8	115
10^{-5}	5	140	424	403	5	1.6	130
10^{-6}	6	161	487	462	6	3.1	158
10^{-7}	6	233	681	656	6	4.2	224
10^{-8}	6	294	865	840	6	5.1	283
10^{-9}	6	401	1174	1149	6	6.1	383
10^{-10}	6	588	1662	1637	6	7.3	534
blended DIFSUB							
10^{-2}	5	59	278	410	18	1.4	100
10^{-3}	7	63	212	366	7	2.8	100
10^{-4}	8	77	247	436	7	3.7	113
10^{-5}	8	99	307	556	7	4.5	145
10^{-6}	10	106	335	596	9	5.1	165
10^{-7}	10	128	394	714	9	6.0	228
10^{-8}	10	157	470	866	9	6.9	273
10^{-9}	11	170	505	928	10	8.2	299
10^{-10}	11	200	579	1076	10	9.5	361

Again there is a significant difference in performance.

7. IMPLICIT DIFFERENTIAL EQUATIONS

One of the advantages of the Nordsieck representation is that a predicted value p'_n for y'_n is available, and hence implicit differential equations $F(t, y, y') = 0$ can be treated without additional complication (cf. Gear [10]). We simply chose Δ_n to satisfy

$$F(t_n, p_n + \ell_0 \Delta_n, p'_n + h^{-1} \ell_1 \Delta_n) = 0. \quad (7.1)$$

The equation $F(t, y, y') = 0$ implicitly defines y' as a function $f(t, y)$ of t and y . Differentiating $F(t, y, f(t, y)) = 0$ with respect to y yields

$$\frac{\partial F}{\partial y} + \frac{\partial F}{\partial y'} \frac{\partial f}{\partial y} = 0,$$

and so the blended formula has

$$\underline{\ell} = \underline{a} + \gamma h K^{-1} J \underline{b}, \quad (7.2)$$

where $J \approx \frac{\partial F}{\partial y}$ and $K \approx \frac{\partial F}{\partial y'}$.

The Newton iteration for (7.1) is

$$\Delta_{(0)} = 0,$$

$$\Delta_{(m+1)} = \Delta_{(m)} - [(\partial F / \partial y)_{(m)} \ell_0 + h^{-1} (\partial F / \partial y')_{(m)} \ell_1]^{-1} F_{(m)}$$

where for a function the subscript (m) denotes evaluation at

$(t, p + \ell_0 \Delta_{(m)}, p' + h^{-1} \ell_1 \Delta_{(m)})$. The modified Newton iteration is

$$\Delta_{(m+1)} = \Delta_{(m)} - [K \ell_1 + h J \ell_0]^{-1} h F_{(m)}$$

or equivalently

$$\Delta_{(m+1)} = \Delta_{(m)} - [\ell_1 + h K^{-1} J \ell_0]^{-1} K^{-1} h F_{(m)}.$$

From §4 we have $\underline{a} = [\beta, 1, \dots]^T$ and $\underline{b} = [1, \alpha, \dots]^T$, and hence from (7.2) we get

$$\ell_1 + h K^{-1} J \ell_0 = 1 + (\gamma \alpha + \beta) h K^{-1} J + \gamma (h K^{-1} J)^2,$$

which we approximate by $(1 + c h K^{-1} J)^2$. The approximate modified Newton iteration is

$$\Delta_{(m+1)} = \Delta_{(m)} - [1 + c h K^{-1} J]^{-2} K^{-1} h F_{(m)}$$

or equivalently

$$\Delta_{(m+1)} = \Delta_{(m)} - [K + c h J]^{-1} K [K + c h J]^{-1} h F_{(m)} .$$

Therefore the blended method can accomodate implicit equations if the corrector step is modified as follows:

$\Delta \leftarrow 0; \tilde{\Delta} \leftarrow 0$

for $m \leftarrow 1, 2, 3$

do { $F \leftarrow F(t, y, y')$
 if desirable then { $K \leftarrow \partial F / \partial y'$
 $cH \leftarrow c \times h$
 $W \leftarrow K + cH \times \partial F / \partial y$
 decompose W
 solve $W \times \tilde{d} = -h \times F$ for $\tilde{d}$
 solve $W \times d = K \times \tilde{d}$ for d
 ...
 }

...

Thus the only additional work would be the computation of K and the multiplication of a vector by K . Also additional storage is required for K .

It is worth noting that the algorithm does not depend on K being nonsingular, and so the algorithm should be suitable for differential-algebraic systems.

8. CONCLUSION

Limited numerical evidence suggests that the blended formulas may be as good as the backward differentiation formulas for stiff problems, better for nonstiff problems, and much better for stiff oscillatory problems. Furthermore they can be incorporated into existing codes such as DIFSUB [9], GEAR Rev. 3 [12], and EPISODE [3], which have benefited from considerable computational experience. It is of interest to note that the AMF-BDF blend may be at least a partial solution to the problem [14, p. 552] of designing a method which automatically selects either an Adams-Moulton formula or a backward differentiation formula for each individual equation in the system.

APPENDIX A. COEFFICIENTS OF THE FORMULAS

$\underline{a}^{(2)}$	$\underline{a}^{(3)}$	$\underline{a}^{(4)}$	$\underline{a}^{(5)}$	$\underline{a}^{(6)}$	$\underline{a}^{(7)}$
$\frac{1}{2}$	$\frac{5}{12}$	$\frac{3}{8}$	$\frac{251}{720}$	$\frac{95}{288}$	$\frac{19087}{60480}$
1	1	1	1	1	1
	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{11}{12}$	$\frac{25}{24}$	$\frac{137}{120}$
		$\frac{1}{6}$	$\frac{1}{3}$	$\frac{35}{72}$	$\frac{5}{8}$
			$\frac{1}{24}$	$\frac{5}{48}$	$\frac{17}{96}$
				$\frac{1}{120}$	$\frac{1}{40}$
					$\frac{1}{720}$

$\underline{a}^{(8)}$	$\underline{a}^{(9)}$	$\underline{a}^{(10)}$	$\underline{a}^{(11)}$	$\underline{a}^{(12)}$
$\frac{5257}{17280}$	$\frac{1070017}{3628800}$	$\frac{25713}{89600}$	$\frac{26842253}{95800320}$	$\frac{4777223}{17418240}$
1	1	1	1	1
$\frac{49}{40}$	$\frac{363}{280}$	$\frac{761}{560}$	$\frac{7129}{5040}$	$\frac{7381}{5040}$
$\frac{203}{270}$	$\frac{469}{540}$	$\frac{29531}{30240}$	$\frac{6515}{6048}$	$\frac{177133}{151200}$
$\frac{49}{192}$	$\frac{967}{2880}$	$\frac{267}{640}$	$\frac{4523}{9072}$	$\frac{84095}{145152}$
$\frac{7}{144}$	$\frac{7}{90}$	$\frac{1069}{9600}$	$\frac{19}{128}$	$\frac{341693}{1814400}$
$\frac{7}{1440}$	$\frac{23}{2160}$	$\frac{3}{160}$	$\frac{3013}{103680}$	$\frac{8591}{207360}$
$\frac{1}{5040}$	$\frac{1}{1260}$	$\frac{13}{6720}$	$\frac{5}{1344}$	$\frac{7513}{1209600}$
	$\frac{1}{40320}$	$\frac{1}{8960}$	$\frac{29}{96768}$	$\frac{121}{193536}$
		$\frac{1}{362880}$	$\frac{1}{72576}$	$\frac{11}{272160}$
			$\frac{1}{3628800}$	$\frac{11}{7257600}$
				$\frac{1}{39916800}$

$\underline{b}^{(1)}$	$\underline{b}^{(2)}$	$\underline{b}^{(3)}$	$\underline{b}^{(4)}$	$\underline{b}^{(5)}$	$\underline{b}^{(6)}$
1	1	1	1	1	1
1	$\frac{3}{2}$	$\frac{11}{6}$	$\frac{25}{12}$	$\frac{137}{60}$	$\frac{49}{20}$
	$\frac{1}{2}$	1	$\frac{35}{24}$	$\frac{15}{8}$	$\frac{203}{90}$
		$\frac{1}{6}$	$\frac{5}{12}$	$\frac{17}{24}$	$\frac{49}{48}$
			$\frac{1}{24}$	$\frac{1}{8}$	$\frac{35}{144}$
				$\frac{1}{120}$	$\frac{7}{240}$
					$\frac{1}{720}$

$\underline{b}^{(7)}$	$\underline{b}^{(8)}$	$\underline{b}^{(9)}$	$\underline{b}^{(10)}$	$\underline{b}^{(11)}$
1	1	1	1	1
$\frac{363}{140}$	$\frac{761}{280}$	$\frac{7129}{2520}$	$\frac{7381}{2520}$	$\frac{83711}{27720}$
$\frac{469}{180}$	$\frac{29531}{10080}$	$\frac{6515}{2016}$	$\frac{177133}{50400}$	$\frac{190553}{50400}$
$\frac{967}{720}$	$\frac{267}{160}$	$\frac{4523}{2268}$	$\frac{84095}{36288}$	$\frac{341747}{129600}$
$\frac{7}{18}$	$\frac{1069}{1920}$	$\frac{95}{128}$	$\frac{341693}{362880}$	$\frac{139381}{120960}$
$\frac{23}{360}$	$\frac{9}{80}$	$\frac{3013}{17280}$	$\frac{8591}{34560}$	$\frac{242537}{725760}$
$\frac{1}{180}$	$\frac{13}{960}$	$\frac{5}{192}$	$\frac{7513}{172800}$	$\frac{1903}{28800}$
$\frac{1}{5040}$	$\frac{1}{1120}$	$\frac{29}{12096}$	$\frac{121}{24192}$	$\frac{10831}{1209600}$
	$\frac{1}{40320}$	$\frac{1}{8064}$	$\frac{11}{30240}$	$\frac{11}{13440}$
		$\frac{1}{362880}$	$\frac{11}{725760}$	$\frac{1}{20736}$
			$\frac{1}{3628800}$	$\frac{1}{604800}$
				$\frac{1}{39916800}$

APPENDIX B. SOURCE CODE FOR BLENDED DIFSUB

```

      IMPLICIT REAL*8(A-H,O-Z)
      COMMON /INTEGER/ IPRDE,NECN,NCINV,NSOLVE
      COMMON /INMAIN/ IQUIT
      CALL UNDERZ('OFF')
5     READ(5,800,END=20) IPRDE,IPOWER
      WRITE(6,1000) IPRDE
1000  FORMAT(////////'1',48X,'P R O B L E M    ',I2)
      WRITE(6,1002)
1002  FORMAT('/'0',3X,'TOLERANCE',2X,'#STEPS',2X,'#FCN',2X,
1      '#DECOMP',2X,'#SOLVE',2X,'MAX ORDER',2X,
2      'MAX ERROR (TC,TF)',3X,'T (MAX ERROR)',3X,
3      'CPU TIME',2X,'KFLAG',2X,'STOP AT IF'/)
      IQUIT = 0
      DO 10 I = 2,IPOWER
          EPS = 10.00**(-I)
          CALL TESTDS(EPS)
          IF ( IQUIT.NE.0 ) GO TO 5
10     CONTINUE
      GO TO 5
20     STOP
800    FORMAT(2I3)
      END

```

```

SUBROUTINE TESTDS(TOL)
IMPLICIT REAL*8(A-H,O-Z)
REAL*4 PW
DIMENSION NDIM(10),TO(10),TF(10),YO(20,10),Y(13,20),ERROR(20),
1      SAVE(16,23),YMAX(20),PW(20,20),OMEGA(20),OMEGA2(20),
2      IF(20)
COMMON /INTEGER/ IPRCB,NFCN,NOINV,NSOLVE
COMMON /IMAIN/ IQUIT
COMMON /INDIF/ MAXNQ
DATA TO / 0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.3*0.00 /
DATA TF / 20.00,1000.00,15.00,15.00,20.00,20.00,1000.00,0.3*0.00 /
DATA YO / 1.00,19*0.00, -1.00,-1.00,-1.00,-1.00,16*0.00,
1      10.00,19*0.00, 2.00,1.00,2.00,17*0.00,
2      1.00,0.00,0.00,1.00,16*0.00, 6*1.00,14*0.00,
3      0.00,-2.00,-1.00,-1.00,16*0.00, 60*0.00 /
DATA NDIM / 1,4,1,3,4,6,4,3*0 /
DATA FOURU / Z334C00C000000000 /
DATA MAXNUM / 1000 /
NFCN = 0
MAXNQ = 0
NSTEP = 0
NSOLVE = 0
NOINV = 0
ERRMAX = 0.00
T MXERR = 0.00
NEQN = NDIM(IPRCB)
T = TO(IPRCB)
ITIME = 0
DO 10 I = 1,NEQN
    Y(1,I) = YO(I,IPRCB)
10  YMAX(I) = DMAX1(1.00,CABS(Y(1,I)))
HMIN = FOURU*DMAX1(DABS(T),CABS(TF(IPRCB)))
H = TCL
JSTART = 0
30 IF ( T.GE.(TF(IPRCB) - HMIN) ) GO TO 70
HMAX = TF(IPRCB) - T
H = DMAX1(HMIN,DMIN1(HMAX,H))
NSTEP = NSTEP + 1
IF ( NSTEP.LE.MAXNUM ) GO TO 35
IQUIT = 1
GO TO 50
35 CALL DIFSUB(NEQN,T,Y,SAVE,H,HMIN,HMAX,TOL,2,YMAX,ERROR,
1      KFLAG,JSTART,11,PW,IP,OMEGA,OMEGA2)
CALL CALERR(T,Y,YMAX,GLBERR)
IF ( GLBERR.LE.ERRMAX ) GO TO 40
ERRMAX = GLBERR
T MXERR = T
40 JSTART = 1
IF ( KFLAG.EQ.1 ) GO TO 30
50 WRITE(6,999) TOL,NSTEP,NFCN,NOINV,NSOLVE,MAXNQ,ERRMAX,T MXERR,
1      ITIME,KFLAG,T
999 FORMAT('0',3X,D9.3,2X,I5,2X,I5,3X,I4,5X,I4,7X,I2,
1      8X,D11.5,7X,D11.5,5X,I6,4X,I2,4X,D10.4)
RETURN
70 WRITE(6,999) TOL,NSTEP,NFCN,NOINV,NSOLVE,MAXNQ,ERRMAX,T MXERR,
1      ITIME
RETURN
END

```

```

SUBROUTINE DECOMP(PW,N,M,IP)
IMPLICIT REAL*8(A-H,O-Z)
REAL*4 PW
DIMENSION PW(M),IP(M)
COMMON /INTEGER/ IPROB,NFCN,NOINV,NSOLVE
NCINV = NOINV + 1
IP(N) = 1
DO 60 K = 1,N
  IF ( K.EQ.N ) GO TO 50
  KP1 = K + 1
  KM1M = (K - 1)*M
  IPIVOT = K
  DO 10 I = KP1,N
    IF ( ABS(PW(I+KM1M)).GT.ABS(PW(IPIVOT+KM1M)) ) IPIVOT = I
10  CONTINUE
  IP(K) = IPIVOT
  IF ( IPIVOT.NE.K ) IP(N) = - IP(N)
  IPKM = IPIVOT + KM1M
  KKM = K + KM1M
  TEMP = PW(IPKM)
  PW(IPKM) = PW(KKM)
  PW(KKM) = TEMP
  IF ( TEMP.EQ.0.D0 ) GO TO 50
  DO 20 I = KP1,N
    IKM = I + KM1M
20  PW(IKM) = - PW(IKM)/TEMP
  DO 40 J = KP1,N
    JM1M = (J - 1)*M
    IPJM = IPIVOT + JM1M
    KJM = K + JM1M
    TEMP = PW(IPJM)
    PW(IPJM) = PW(KJM)
    PW(KJM) = TEMP
    IF ( TEMP.EQ.0.D0 ) GO TO 40
    DO 30 I = KP1,N
      IJM = I + JM1M
30  PW(IJM) = PW(IJM) + PW(I+KM1M)*TEMP
40  CONTINUE
50  IF ( PW(KKM).EQ.0.C ) IP(N) = 0
60  CCNTINUE
RETURN
END

```

```

SUBROUTINE SOLVE(PW,N,M,IP,B)
IMPLICIT REAL*8 (A-H,C-Z)
REAL*4 PW
DIMENSION PW(M),IP(M),B(M)
COMMON /INTGER/ IPROB,NFCN,NCINV,NSOLVE
NSOLVE = NSOLVE + 1
IF ( N.EQ.1 ) GO TO 30
NM1 = N - 1
DO 10 K = 1,NM1
  KP1 = K + 1
  KM1M = (K - 1)*M
  IPIVOT = IP(K)
  TEMP = B(IPIVOT)
  B(IPIVOT) = B(K)
  E(K) = TEMP
  DO 10 I = KP1,N
10    E(I) = B(I) + PW(I+KM1M)*TEMP
  DO 20 KB = 1,NM1
    KM1 = N - KB
    K = KM1 + 1
    KM1M = (K - 1)*M
    B(K) = B(K)/PW(K+KM1M)
    TEMP = - E(K)
    DO 20 I = 1,KM1
20    E(I) = E(I) + PW(I+KM1M)*TEMP
30 E(1) = B(1)/PW(1)
RETURN
END

```



```

SUBROUTINE DIFFUN(T,Y,DY)
IMPLICIT REAL*8(A-H,C-Z)
DIMENSION Y(13,20),DY(20)
COMMON /INTEGER/ IPROB,NFCN,NOINV,NSOLVE
NFCN = NFCN + 1
GO TO (10,20,30,40,50,60,70,80,90,100),IPROB
10 DY(1) = - Y(1,1)
RETURN
20 ZTEMP = .500*(Y(1,1) + Y(1,2) + Y(1,3) + Y(1,4))
Z1 = ZTEMP - Y(1,1)
Z2 = ZTEMP - Y(1,2)
Z3 = ZTEMP - Y(1,3)
Z4 = ZTEMP - Y(1,4)
Z1 = Z1*(Z1 - 1000.00)
Z2 = Z2*(Z2 - 800.00)
Z3 = Z3*(Z3 + 10.00)
Z4 = Z4*(Z4 - .00100)
ZTEMP = .500*(Z1 + Z2 + Z3 + Z4)
CY(1) = ZTEMP - Z1
CY(2) = ZTEMP - Z2
CY(3) = ZTEMP - Z3
CY(4) = ZTEMP - Z4
RETURN
30 CY(1) = - 200.00*Y(1,1) + 2000.00 - (1991.00 +
1 199.00*T)*DEXP(-T)
RETURN
40 CY(1) = - .100*Y(1,1) - 49.900*Y(1,2)
CY(2) = - 50.00*Y(1,2)
CY(3) = 70.00*Y(1,2) - 120.00*Y(1,3)
RETURN
50 CY(1) = Y(1,3)
CY(2) = Y(1,4)
CY(4) = (Y(1,1)*Y(1,1) + Y(1,2)*Y(1,2))**.5
CY(3) = - Y(1,1)/DY(4)
CY(4) = - Y(1,2)/DY(4)
RETURN
60 CY(1) = - 10.00*Y(1,1) + 100.00*Y(1,2)
CY(2) = - 100.00*Y(1,1) - 10.00*Y(1,2)
CY(3) = - 4.00*Y(1,3)
CY(4) = - Y(1,4)
CY(5) = - .500*Y(1,5)
CY(6) = - .100*Y(1,6)
RETURN
70 ZTEMP = .500*(Y(1,1) + Y(1,2) + Y(1,3) + Y(1,4))
Z1 = ZTEMP - Y(1,1)
Z2 = ZTEMP - Y(1,2)
Z3 = ZTEMP - Y(1,3)
Z4 = ZTEMP - Y(1,4)
W1 = .500*(Z1*Z1 - Z2*Z2)
W2 = Z1*Z2
T1 = Z1
Z1 = 10.00*T1 + 10.00*Z2 + W1
Z2 = - 10.00*T1 + 10.00*Z2 + W2
Z3 = Z3*(Z3 - 1000.00)
Z4 = Z4*(Z4 - .0100)
ZTEMP = .500*(Z1 + Z2 + Z3 + Z4)
CY(1) = ZTEMP - Z1
CY(2) = ZTEMP - Z2
CY(3) = ZTEMP - Z3

```

```
      CY(4) = ZTEMP - Z4  
      RETURN  
80  RETURN  
90  RETURN  
100 RETURN  
      END
```



```

SUBROUTINE CALERR(T,Y,YMAX,GLEBERR)
IMPLICIT REAL*8(A-H,O-Z)
DIMENSION Y(13,20),YMAX(20)
COMMON /INTEGER/ IPROB,NFCN,NOINV,NSOLVE
GO TO (10,20,30,40,50,60,70,80,90,100),IPROB
10 GLEBERR = DABS((Y(1,1) - DEXP(-T))/YMAX(1))
RETURN
20 T1 = DEXP(- 1000.00*T)
T2 = DEXP(- 800.00*T)
T3 = DEXP(- 10.00*T)
T4 = DEXP(- .00100*T)
Z1 = 1000.00*T1/(T1 - 1001.00)
Z2 = 800.00*T2/(T2 - 801.00)
Z3 = - 10.00/(1.00 + 9.00*T3)
Z4 = .00100*T4/(T4 - 1.00100)
ZTEMP = .500*(Z1 + Z2 + Z3 + Z4)
GLEBERR = ((Y(1,1) - ZTEMP + Z1)/YMAX(1))**2
1 + ((Y(1,2) - ZTEMP + Z2)/YMAX(2))**2
2 + ((Y(1,3) - ZTEMP + Z3)/YMAX(3))**2
3 + ((Y(1,4) - ZTEMP + Z4)/YMAX(4))**2
GLEBERR = DSQRT(GLEBERR)
RETURN
30 GLEBERR = DABS((Y(1,1) - 10.00 + (10.00 + T)*DEXP(-T)
1 - 10.00*DEXP(- 200.00*T))/YMAX(1))
RETURN
40 T1 = DEXP(- 50.00*T)
GLEBERR = ((Y(1,1) - DEXP(- .100*T) - T1)/YMAX(1))**2
1 + ((Y(1,2) - T1)/YMAX(2))**2
2 + ((Y(1,3) - T1 - DEXP(- 120.00*T))/YMAX(3))**2
GLEBERR = DSQRT(GLEBERR)
RETURN
50 CCS T = DCCS(T)
SIN T = DSIN(T)
GLEBERR = ((Y(1,1) - CCS T)/YMAX(1))**2
1 + ((Y(1,2) - SIN T)/YMAX(2))**2
2 + ((Y(1,3) + SIN T)/YMAX(3))**2
3 + ((Y(1,4) - CCS T)/YMAX(4))**2
GLEBERR = DSQRT(GLEBERR)
RETURN
60 T1 = DCOS(100.00*T)
T2 = DSIN(100.00*T)
T3 = DEXP(- 10.00*T)
GLEBERR = ((Y(1,1) - T3*(T1 + T2))/YMAX(1))**2
1 + ((Y(1,2) - T3*(T1 - T2))/YMAX(2))**2
2 + ((Y(1,3) - DEXP(- 4.00*T))/YMAX(3))**2
3 + ((Y(1,4) - DEXP(-T))/YMAX(4))**2
4 + ((Y(1,5) - DEXP(- .500*T))/YMAX(5))**2
5 + ((Y(1,6) - DEXP(- .100*T))/YMAX(6))**2
GLEBERR = DSQRT(GLEBERR)
RETURN
70 T1 = DEXP(- 10.00*T)
T2 = DEXP(- 1000.00*T)
T3 = DEXP(- .0100*T)
ECCSBT = DCCS(-10.00*T)*T1
ESINBT = DSIN(-10.00*T)*T1
DENCM = (1.00 + 9.00*ECCSBT - 10.00*ESINBT)**2
1 + (- 9.00*ESINBT - 10.00*ECCSBT)**2
Z1 = - 20.00*(1.00 + 19.00*ECCSBT - ESINBT)/DENCM
Z2 = 20.00*(1.00 - ECCSBT - 19.00*ESINBT)/DENCM

```

```

      Z3 = 1000.D0*T2/(T2 - 1001.D0)
      Z4 = .0100*I3/(I3 - 1.01C0)
      ZTEMP = .5D0*(Z1 + Z2 + Z3 + Z4)
      GLBERR = ((Y(1,1) - ZTEMP + Z1)/YMAX(1))**2
1      + ((Y(1,2) - ZTEMP + Z2)/YMAX(2))**2
2      + ((Y(1,3) - ZTEMP + Z3)/YMAX(3))**2
3      + ((Y(1,4) - ZTEMP + Z4)/YMAX(4))**2
      GLBERR = DSCRT(GLBERR)
      RETURN
80 RETURN
90 RETURN
100 RETURN
      END

```

```

SUBROUTINE PEDERV(T,Y,PW,N)
  IMPLICIT REAL*8(A-H,O-Z)
  REAL*4 PW(N,N)
  DIMENSION Y(13,20)
  RETURN
  END

```

SUBROUTINE DIFSUE(N,T,Y,SAVE,H,HMIN,HMAX,EPS,MF,YMAX,ERROR,KFLAG,

1 JSTART,MAXDER,PW,IF,OMEGA,OMEGA2)

IMPLICIT REAL*8 (A-H,Q-Z)

```

C*****
C*
C* THIS SUBROUTINE INTEGRATES A SET OF N ORDINARY DIFFERENTIAL FIRST
C* ORDER EQUATIONS OVER ONE STEP OF LENGTH H AT EACH CALL. H CAN BE
C* SPECIFIED BY THE USER FOR EACH STEP, BUT IT MAY BE INCREASED OR
C* DECREASED BY DIFSUE WITHIN THE RANGE HMIN TO HMAX IN ORDER TO
C* ACHIEVE AS LARGE A STEP AS POSSIBLE WHILE NOT COMMITTING A SINGLE
C* STEP ERROR WHICH IS LARGER THAN EPS IN THE L-2 NORM, WHERE EACH
C* COMPONENT OF THE ERROR IS DIVIDED BY THE COMPONENTS OF YMAX.
C*
C* THE PROGRAM REQUIRES THREE SUBROUTINES NAMED
C*   DIFFUN(T,Y,DY)
C*   MATINV(PW,N,M,J)
C*   PEDERV(T,Y,PW,M)
C* THE FIRST, DIFFUN, EVALUATES THE DERIVATIVES OF THE DEPENDENT
C* VARIABLES STORED IN Y(1,I) FOR I = 1 TO N, AND STORES THE
C* DERIVATIVES IN THE ARRAY DY. THE SECOND IS CALLED ONLY IF THE
C* METHOD FLAG MF IS SET TO 1 OR 2 FOR STIFF METHODS. IT MUST INVERT
C* THE N BY N MATRIX STORED IN THE ARRAY PW(M,M). IF THE INVERSION IS
C* SUCCESSFUL, J SHOULD BE SET TO 1, OTHERWISE IT SHOULD BE SET TO -1.
C* PEDERV IS USED ONLY IF MF IS 1, AND COMPUTES THE PARTIAL
C* DERIVATIVES OF THE DIFFERENTIAL EQUATIONS AS DESCRIBED UNDER THE
C* MF PARAMETER.
C*
C* THE PROGRAM USES DOUBLE PRECISION ARITHMETIC FOR ALL FLOATING
C* POINT VARIABLES EXCEPT THOSE STARTING WITH P. THE FORMER ARE
C* SINGLE PRECISION TO SAVE TIME AND SPACE.
C*
C* THE TEMPORARY STORAGE SPACE IS PROVIDED BY THE CALLER IN THE
C* SINGLE PRECISION ARRAY PW AND THE DOUBLE PRECISION ARRAY SAVE.
C* IEY0331 COMMENTS DELETED *****
C*   DIMENSION Y(13,N),YMAX(N),SAVE(16,N),ERROR(N),PW(N),OMEGA(N),
C*   1 OMEGA2(N),IF(N),
C*   2 A(13),B(12),C(11),GAMMA(11),PERTST(12,2,3)
C*****
C* THE COEFFICIENTS IN PERTST ARE USED IN SELECTING THE STEP AND
C* ORDER, THEREFORE ONLY ABOUT ONE PERCENT ACCURACY IS NEEDED.
C*****
C* DATA PERTST
C* 1 / 2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0,11.0,12.0,+13.0,
C* 2 2.0,12.0,24.0,37.89,53.33,70.08,87.97,106.9,126.7,147.4,
C* 2 168.8,191.0,
C* 3 3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0,11.0,12.0,+13.0,+14.0,
C* 4 12.0,24.0,37.89,53.33,70.08,87.97,106.9,126.7,147.4,168.8,
C* 4 191.0,+213.8,
C* 5 +1.0,1.0,.5,.1667,.04167,.008333,.001389,.0001984,
C* 5 .00002480,.000002756,.0000002756,+.00000002505,
C* 6 +1.0,1.0,2.0,1.0,.3158,.07407,.01390,.002182,
C* 6 .0002945,.00003492,.000003692,.0000003524 /
C* DATA A(2) / -1.0D0 /
C* DATA B(1) / -1.0D0 /
C* DATA GAMMA / .08578644D0,.125D0,.1218908D0,.1284997D0,.1087264D0,
C* 1 .09625961D0,.08754865D0,.08105623D0,.07599874D0,
C* 2 .07192936D0,.06857227D0 /
C* DATA C / .2928932D0,.3374973D0,.3335427D0,.3427329D0,.3169058D0,
C* 1 .2992971D0,.2862392D0,.2760327D0,.2677630D0,.2608834D0,

```


2

.2550426D0 /

```

CXXXXXXXXXXXXPLEASE REMOVEXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
COMMON /INDIF/ MAXNQ
CXXXXXXXXXXXXDOES NOT BELONG HEREXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  IRET = 1
  KFLAG = 1
  IF (JSTART.LE.0) GO TO 140
C*****
C* BEGIN BY SAVING INFORMATION FOR POSSIBLE RESTARTS AND CHANGING
C* H BY THE FACTOR R IF THE CALLER HAS CHANGED H. ALL VARIABLES
C* DEPENDENT ON H MUST ALSO BE CHANGED.
C* E IS A COMPARISON FOR ERRORS OF THE CURRENT ORDER NO. EUP IS
C* TO TEST FOR INCREASING THE ORDER, EDWN FOR DECREASING THE ORDER.
C* HNEW IS THE STEP SIZE THAT WAS USED ON THE LAST CALL.
C*****
100 DO 110 I = 1,N
    DO 110 J = 1,K
110     SAVE(J,I) = Y(J,I)
    HOLD = HNEW
    RACUM = 1.0
    IF (H.EQ.HOLD) GO TO 130
120 RACUM = H/HOLD
    IRET1 = 1
    GO TO 750
130 NGOLD = NQ
    TOLD = T
    IF (JSTART.GT.0) GO TO 250
    GO TO 170
140 IF (JSTART.EQ.-1) GO TO 160
C*****
C* ON THE FIRST CALL, THE ORDER IS SET TO 1 AND THE INITIAL
C* DERIVATIVES ARE CALCULATED.
C*****
    NQ = 1
    N3 = N
    N1 = N*16
    N2 = N1 + 1
    N4 = N**2
    N5 = N1 + N
    N6 = N5 + 1
    CALL DIFFUN(T,Y,SAVE(N2,1))
    DO 150 I = 1,N
        N11 = N1 + I
150     Y(2,I) = SAVE(N11,1)*H
    FNEW = H
    K = 2
    GO TO 100
C*****
C* REPEAT LAST STEP BY RESTORING SAVED INFORMATION.
C*****
160 IF (NQ.EQ.NGOLD) JSTART = 1
    T = TOLD
    NQ = NGOLD
    K = NQ + 1
    GO TO 120
C*****
C* SET THE COEFFICIENTS THAT DETERMINE THE ORDER AND THE METHOD
C* TYPE. CHECK FOR EXCESSIVE ORDER. THE LAST TWO STATEMENTS OF
C* THIS SECTION SET INEVAL .GT.0 IF PW IS TO BE RE-EVALUATED

```

```

C* BECAUSE OF THE ORDER CHANGE, AND THEN REPEAT THE INTEGRATION
C* STEP IF IT HAS NOT YET BEEN DONE (IRET = 1) OR SKIP TO A FINAL
C* SCALING BEFORE EXIT IF IT HAS BEEN COMPLETED (IRET = 2).
C*****
170  IF (MF.EQ.0) GO TO 180
      IF (NG.GT.11) GO TO 190
      GO TO (221,222,223,224,225,226,227,228,229,230,231),NQ
180  IF (NG.GT.12) GO TO 190
      GO TO (201,202,203,204,205,206,207,208,209,210,211,212),NQ
190  KFLAG = -2
      RETURN
C*****
C* THE FOLLOWING COEFFICIENTS SHOULD BE DEFINED TO THE MAXIMUM
C* ACCURACY PERMITTED BY THE MACHINE. THEY ARE, IN THE ORDER USED..
C*
C* -1
C* -1/2,-1/2
C* -5/12,-3/4,-1/6
C* -3/8,-11/12,-1/3,-1/24
C* -251/720,-25/24,-35/72,-5/48,-1/120
C* -95/288,-137/120,-5/8,-17/96,-1/40,-1/720
C* -19087/60480,-49/40,-203/270,-49/192,-7/144,-7/1440,-1/5040
C*
C* -1
C* -2/3,-1/3
C* -6/11,-6/11,-1/11
C* -12/25,-7/10,-1/5,-1/50
C* -120/274,-225/274,-85/274,-15/274,-1/274
C* -180/441,-58/63,-15/36,-25/252,-3/252,-1/1764
C*****
201  A(1) = - .5D0
      GO TO 235
202  A(1) = - .41666666666666667D0
      A(3) = - .5D0
      GO TO 235
203  A(1) = - .375D0
      A(3) = - .75D0
      A(4) = - .16666666666666667D0
      GO TO 235
204  A(1) = - .3486111111111111D0
      A(3) = - .91666666666666667D0
      A(4) = - .3333333333333333D0
      A(5) = - .04166666666666667D0
      GO TO 235
205  A(1) = - .3298611111111111D0
      A(3) = - 1.0416666666666667D0
      A(4) = - .4861111111111111D0
      A(5) = - .10416666666666667D0
      A(6) = - .008333333333333333D0
      GO TO 235
206  A(1) = - .3155919312169312D0
      A(3) = - 1.1416666666666667D0
      A(4) = - .625D0
      A(5) = - .1770833333333333D0
      A(6) = - .025D0
      A(7) = - .00138888888888889D0
      GO TO 235
207  A(1) = - .3042245370370370D0
      A(3) = - 1.225D0

```

A(4) = - .7518518518518519D0
 A(5) = - .2552083333333333D0
 A(6) = - .0486111111111111D0
 A(7) = - .0C48611111111111D0
 A(8) = - .0001984126984126984D0

GC TO 235

208 A(1) = - .2548680004409171D0
 A(3) = - 1.296428571428485D0
 A(4) = - .8685185185185185D0
 A(5) = - .3357638888888888D0
 A(6) = - .0777777777777777D0
 A(7) = - .01064814814814815D0
 A(8) = - .0007936507936507937D0
 A(9) = - .0000248015873015873D0

GC TO 235

209 A(1) = - .2869754464285714D0
 A(3) = - 1.35892857142857D0
 A(4) = - .976554232804233D0
 A(5) = - .4171875D0
 A(6) = - .1113541666666667D0
 A(7) = - .01875D0
 A(8) = - .001934523809523809D0
 A(9) = - .000111607142857143D0
 A(10) = - .C0C0C275573192239859D0

GC TC 235

210 A(1) = - .280189564439367D0
 A(3) = - 1.41448412698413D0
 A(4) = - 1.C7721560846561D0
 A(5) = - .498567C1940C353D0
 A(6) = - .1484375D0
 A(7) = - .0290605709876543D0
 A(8) = - .0037202380952381D0
 A(9) = - .000299685846560847D0
 A(10) = - .C0C0137786596119929D0
 A(11) = - .C0C00C27557319223986D0

GC TC 235

211 A(1) = - .274265540031599D0
 A(3) = - 1.46448412698413D0
 A(4) = - 1.17151455026455D0
 A(5) = - .579358190035273D0
 A(6) = - .188322861552028D0
 A(7) = - .0414303626543210D0
 A(8) = - .0062111447989418D0
 A(9) = - .C00625206679894180D0
 A(10) = - .00C0404174015285126D0
 A(11) = - .00C00151565255731922D0
 A(12) = - .C0C000025C521C83854417D0

GC TO 235

212 A(1) = - .2690288467736492D0
 A(3) = - 1.50993867243867D0
 A(4) = - 1.26027116402116D0
 A(5) = - .659234182098765D0
 A(6) = - .230458002645503D0
 A(7) = - .0556972461C52322D0
 A(8) = - .00943948412698413D0
 A(9) = - .00111927496693122D0
 A(10) = - .00C0909391534391534D0
 A(11) = - .C0C00482253C86415753D0
 A(12) = - .C0C00C150312650312650D0

A(13) = - .000000020E767569878681D0

GC TO 235

221 E(2) = - 1.00

GC TO 201

222 E(2) = - 1.500

E(3) = - .500

GC TO 202

223 E(2) = - 1.833333333333333D0

E(3) = - 1.00

E(4) = - .1666666666666667D0

GC TO 203

224 E(2) = - 2.083333333333333D0

E(3) = - 1.458333333333333D0

E(4) = - .4166666666666667D0

E(5) = - .0416666666666667D0

GC TO 204

225 E(2) = - 2.283333333333333D0

E(3) = - 1.875D0

E(4) = - .708333333333333D0

E(5) = - .125D0

E(6) = - .00833333333333333D0

GC TO 205

226 E(2) = - 2.45D0

E(3) = - 2.2555555555555556D0

E(4) = - 1.020833333333333D0

E(5) = - .2430555555555556D0

E(6) = - .02916666666666667D0

E(7) = - .001388888888888889D0

GC TO 206

227 E(2) = - 2.59285714285714D0

E(3) = - 2.6055555555555556D0

E(4) = - 1.3430555555555556D0

E(5) = - .3888888888888889D0

E(6) = - .0638888888888889D0

E(7) = - .005555555555555556D0

E(8) = - .000198412698412698D0

GC TO 207

228 E(2) = - 2.71785714285714D0

E(3) = - 2.92966269841270D0

E(4) = - 1.66875D0

E(5) = - .5567708333333333D0

E(6) = - .1125D0

E(7) = - .01354166666666667D0

E(8) = - .000892857142857143D0

E(9) = - .0000248015873015873D0

GC TO 208

229 E(2) = - 2.82896825396825D0

E(3) = - 3.23164682539683D0

E(4) = - 1.99426807760141D0

E(5) = - .7421875D0

E(6) = - .174363425925926D0

E(7) = - .02604166666666667D0

E(8) = - .00239748677248677D0

E(9) = - .000124007936507937D0

E(10) = - .0000275573192239859D0

GC TO 209

230 E(2) = - 2.92896825396825D0

E(3) = - 3.51454365079365D0

E(4) = - 2.31743276014109D0


```

E(5) = - .941614307760141D0
E(6) = - .248582175925926D0
E(7) = - .0434780092592593D0
E(8) = - .00500165343915344D0
E(9) = - .000363756613756614D0
E(10) = - .0000151565255731922D0
E(11) = - .000000275573192239659D0
GO TO 210

```

```

231 E(2) = - 3.01987734487735D0
E(3) = - 3.78081349206349D0
E(4) = - 2.63693672839506D0
E(5) = - 1.15229001322751D0
E(6) = - .334183476631393D0
E(7) = - .0660763888888889D0
E(8) = - .00895419973544574D0
E(9) = - .000818452380952381D0
E(10) = - .0000482253066419753D0
E(11) = - .00000165343915343915D0
E(12) = - .0000000250521083854417D0
GO TO 211

```

```

235 K = NG+1
IDCUB = K
MTYP = (4 - MF)/2
ENG2 = .5/FLOAT(NG + 1)
ENG3 = .5/FLOAT(NG + 2)
ENQ1 = 0.5/FLOAT(NQ)
PEPSH = EPS
EUP = (PERTST(NQ,MTYP,2)*PEPSH)**2
E = (PERTST(NQ,MTYP,1)*PEPSH)**2
EDWN = (PERTST(NQ,MTYP,3)*PEPSH)**2
IF (EDWN.EQ.0) GO TO 780
ENC = EPS*ENQ3/DFLOAT(N)

```

```

240 INEVAL = MF
GO TO ( 250 , 680 ),IRET

```

```

*****
C* THIS SECTION COMPUTES THE PREDICTED VALUES BY EFFECTIVELY
C* MULTIPLYING THE SAVED INFORMATION BY THE PASCAL TRIANGLE
C* MATRIX.

```

```

*****

```

```

250 T = T + H
DO 260 J = 2,K
  DO 260 J1 = J,K
    J2 = K - J1 + J - 1
    DO 260 I = 1,N

```

```

260 Y(J2,I) = Y(J2,I) + Y(J2+1,I)

```

```

*****

```

```

C* UP TO 3 CORRECTOR ITERATIONS ARE TAKEN. CONVERGENCE IS TESTED
C* BY REQUIRING CHANGES TO BE LESS THAN END WHICH IS DEPENDENT ON
C* THE ERROR TEST CONSTANT.
C* THE SUM OF THE CORRECTIONS IS ACCUMULATED IN THE ARRAY
C* ERRCR(I). IT IS EQUAL TO THE K-TH DERIVATIVE OF Y MULTIPLIED
C* BY H**K/(FACTORIAL(K-1)*A(K)), AND IS THEREFORE PROPORTIONAL
C* TO THE ACTUAL ERRORS TO THE LOWEST POWER OF H PRESENT. (H**K)

```

```

*****

```

```

DO 270 I = 1,N
  CMEGA(I) = C.DC
  CMEGA2(I) = 0.D0

```

```

270 SAVE(15,I) = 0.D0
DO 430 L = 1,3

```

CALL DIFFUN (T,Y,SAVE(N2,1))

```

*****
C* IF THERE HAS BEEN A CHANGE OF ORDER OR THERE HAS BEEN TROUBLE *
C* WITH CONVERGENCE, PW IS RE-EVALUATED PRIOR TO STARTING THE *
C* CORRECTOR ITERATION IN THE CASE OF STIFF METHODS. IWEVAL IS *
C* THEN SET TO -1 AS AN INDICATOR THAT IT HAS BEEN DONE. *
*****
      IF (IWEVAL.LT.1) GO TO 350
      XMNS HC = - H*(NQ)
      IF (MF.EQ.2) GO TO 310
      CALL PEDERV(T,Y,PW,N3)
      DO 280 I = 1,N4
280      PW(I) = XMNS HC*PW(I)
290      N11 = N3 + 1
      N12 = N*N11 - N3
      DO 300 I = 1,N12,N11
300      PW(I) = 1.0 + PW(I)
      IWEVAL = -1
      CALL DECCMP(PW,N,N3,IP)
      IF (IP(N).NE.0) GO TO 350
      GO TO 440
*****
C* EVALUATE THE JACOBIAN INTO PW BY NUMERICAL DIFFERENCING. R IS THE *
C* CHANGE MADE TO THE ELEMENT OF Y. IT IS EPS RELATIVE TO Y WITH *
C* A MINIMUM OF EPS**2. *
*****
310      DO 320 I = 1,N
320      SAVE(14,I) = Y(1,I)
      DO 340 J = 1,N
      R = EPS*DMAX1(EPS,CABS(SAVE(14,J)))
      Y(1,J) = Y(1,J) + R
      C = XMNS HC/R
      CALL DIFFUN(T,Y,SAVE(N6,1))
      DO 330 I = 1,N
      N11 = I + (J-1)*N3
      N12 = N5 + I
      N13 = N1 + I
330      PW(N11) = (SAVE(N12,1) - SAVE(N13,1))*D
340      Y(1,J) = SAVE(14,J)
      GO TO 290
350      IF (MF.NE.0) GO TO 370
      DO 360 I = 1,N
      N11 = N1 + I
360      SAVE(14,I) = Y(2,I) - SAVE(N11,1)*H
      GO TO 410
370      DO 380 I = 1,N
      N11 = N5 + I
      N12 = N1 + I
380      SAVE(N11,1) = Y(2,I) - SAVE(N12,1)*H
      CALL SOLVE(PW,N,N3,IF,SAVE(N6,1))
      DO 400 I = 1,N
      N12 = N5 + I
400      SAVE(15,I) = SAVE(N12,1)
      CALL SOLVE(PW,N,N3,IF,SAVE(N6,1))
      R = GAMMA(NQ)*H/XMNS HC
      DO 403 I = 1,N
      N12 = N5 + I
      SAVE(14,I) = SAVE(N12,1)
403      SAVE(15,I) = R*(SAVE(14,I) - SAVE(15,I))

```

```

410      NT = N
C*****
C* CORRECT AND SEE IF ALL CHANGES ARE LESS THAN BND RELATIVE TO YMAX.
C* IF SO, THE CORRECTOR IS SAID TO HAVE CONVERGED.
C*****
      DO 420 I = 1,N
        Y(1,I) = Y(1,I) + A(1)*SAVE(14,I) - SAVE(15,I)
        Y(2,I) = Y(2,I) - SAVE(14,I) + B(2)*SAVE(15,I)
        OMEGA(I) = OMEGA(I) + SAVE(14,I)
        CMEGA2(I) = CMEGA2(I) + SAVE(15,I)
        IF (DABS(SAVE(14,I)+SAVE(15,I)).LE.(BND*YMAX(I))) NT = NT - 1
420      CONTINUE
      IF (NT.LE.0) GC TO 490
430      CONTINUE
C*****
C* THE CORRECTOR ITERATION FAILED TO CONVERGE IN 3 TRIES. VARIOUS
C* POSSIBILITIES ARE CHECKED FOR. IF H IS ALREADY HMIN AND
C* THIS IS EITHER ADAMS METHOD OR THE STIFF METHOD IN WHICH THE
C* MATRIX PW HAS ALREADY BEEN RE-EVALUATED, A NO CONVERGENCE EXIT
C* IS TAKEN. OTHERWISE THE MATRIX PW IS RE-EVALUATED AND/OR THE
C* STEP IS REDUCED TO TRY AND GET CONVERGENCE.
C*****
440      T = T - H
      IF ((F.LE.(HMIN*1.00001)).AND.((IWEVAL - MTYP).LT.-1)) GO TO 460
      IF ((MF.EQ.0).OR.(IWEVAL.NE.0)) RACUM = RACUM*0.2500
      IWEVAL = MF
      IRET1 = 2
      GC TO 750
460      KFLAG = -3
470      DO 480 I = 1,N
        DO 480 J = 1,K
480          Y(J,I) = SAVE(J,I)
      F = FOLD
      NQ = NGOLD
      JSTART = NQ
      RETURN
C*****
C* THE CORRECTOR CONVERGED AND CONTROL IS PASSED TO STATEMENT 520
C* IF THE ERROR TEST IS O.K., AND TO 540 OTHERWISE.
C* IF THE STEP IS O.K. IT IS ACCEPTED. IF IDOUB HAS BEEN REDUCED
C* TO ONE, A TEST IS MADE TO SEE IF THE STEP CAN BE INCREASED
C* AT THE CURRENT ORDER OR BY GOING TO ONE HIGHER OR ONE LOWER.
C* SUCH A CHANGE IS ONLY MADE IF THE STEP CAN BE INCREASED BY AT
C* LEAST 1.1. IF NO CHANGE IS POSSIBLE IDOUB IS SET TO 10 TO
C* PREVENT FURTHER TESTING FOR 10 STEPS.
C* IF A CHANGE IS POSSIBLE, IT IS MADE AND IDOUB IS SET TO
C* NQ + 1 TO PREVENT FURTHER TESTING FOR THAT NUMBER OF STEPS.
C* IF THE ERROR WAS TOO LARGE, THE OPTIMUM STEP SIZE FOR THIS OR
C* LOWER ORDER IS COMPUTED, AND THE STEP RETRIED. IF IT SHOULD
C* FAIL TWICE MORE IT IS AN INDICATION THAT THE DERIVATIVES THAT
C* HAVE ACCUMULATED IN THE Y ARRAY HAVE ERRORS OF THE WRONG ORDER
C* SO THE FIRST DERIVATIVES ARE RECOMPUTED AND THE ORDER IS SET
C* TO 1.
C*****
490      D = 0.0
      CC 500 I = 1,N
      ERROR(I) = OMEGA(I) + CMEGA2(I)
500      D = D + (ERROR(I)/YMAX(I))**2
      IWEVAL = 0

```


IF (C.GT.E) GO TO 540
IF (K.LT.3) GO TO 520

C* COMPLETE THE CORRECTION OF THE HIGHER ORDER DERIVATIVES AFTER A *
C* SUCCESSFUL STEP. *

DO 510 J = 3,K

DO 510 I = 1,N

510 Y(J,I) = Y(J,I) + A(J)*CMEGA(I) + B(J)*CMEGA2(I)

520 KFLAG = +1

FNEW = H

CXXXXXXXXXXXXPLEASE REMOVEXX
MAXNG = MAXC(MAXNG,K)

CXXXXXXXXXXXXDOES NOT BELONG HEREXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

IF (IDCUB.LE.1) GO TO 550

IDCUB = IDCUB - 1

IF (IDCUB.GT.1) GO TO 700

DO 530 I = 1,N

530 SAVE(16,I) = ERROR(I)

GO TO 700

C* REDUCE THE FAILURE FLAG COUNT TO CHECK FOR MULTIPLE FAILURES. *
C* RESTORE T TO ITS ORIGINAL VALUE AND TRY AGAIN UNLESS THERE HAVE *
C* THREE FAILURES. IN THAT CASE THE DERIVATIVES ARE ASSUMED TO HAVE *
C* ACCUMULATED ERRORS SO A RESTART FROM THE CURRENT VALUES OF Y IS *
C* TRIED. *

540 KFLAG = KFLAG - 2

IF (H.LE.(HMIN*1.00001)) GO TO 740

T = TOLD

IF (KFLAG.LE.-5) GO TO 720

C* PR1, PR2, AND PR3 WILL CONTAIN THE AMOUNTS BY WHICH THE STEP SIZE *
C* COULD BE DIVIDED AT ORDER ONE LOWER, AT THIS ORDER, AND AT ORDER *
C* ONE HIGHER RESPECTIVELY. *

550 PR2 = (D/E)**ENQ2*1.2

PR3 = 1.E+20

IF ((NQ.GE.MAXDER).OR.(KFLAG.LE.-1)) GO TO 570

D = 0.0

DO 560 I = 1,N

560 D = D + ((ERROR(I) - SAVE(16,I))/YMAX(I))**2

PR3 = (D/EUP)**ENC3*1.4

570 PR1 = 1.E+20

IF (NQ.LE.1) GO TO 590

D = 0.0

DO 580 I = 1,N

580 D = D + (Y(K,I)/YMAX(I))**2

PR1 = (D/EDWN)**ENQ1*1.3

590 CONTINUE

IF (PR2.LE.PR3) GO TO 650

IF (PR3.LT.PR1) GO TO 660

600 R = 1.0/AMAX1(PR1,1.E-4)

NEWQ = NQ - 1

610 IDCUB = 10

IF ((KFLAG.EQ.1).AND.(R.LT.(1.1))) GO TO 700

IF (NEWQ.LE.NQ) GO TO 630

C* COMPUTE ONE ADDITIONAL SCALED DERIVATIVE IF ORDER IS INCREASED, *

C*****

```

      DC 620 I = 1,N
620   Y(NEWQ+1,I) = ERROR(I)*A(K)/DFLOAT(K)
630   K = NEWQ + 1
      IF (KFLAG.EQ.1) GO TO 670
      RACUM = RACUM*R
      IRET1 = 3
      GO TO 750
640   IF (NEWQ.EQ.NQ) GO TO 250
      NQ = NEWQ
      GO TO 170
650   IF (PR2.GT.PR1) GO TO 600
      NEWQ = NQ
      R = 1.0/AMAX1(PR2,1.E-4)
      GO TO 610
660   R = 1.0/AMAX1(PR3,1.E-4)
      NEWQ = NQ + 1
      GO TO 610
670   IRET = 2
      R = DMIN1(R,HMAX/DAES(H))
      H = H*R
      HNEW = H
      IF (NQ.EQ.NEWQ) GO TO 680
      NQ = NEWQ
      GO TO 170
680   R1 = 1.0
      DO 690 J = 2,K
         R1 = R1*R
      DO 690 I = 1,N
         Y(J,I) = Y(J,I)*R1
      IDOUB = K
700   DO 710 I = 1,N
710   YMAX(I) = DMAX1(YMAX(I),DAES(Y(1,I)))
      JSTART = NQ
      RETURN
720   IF (NG.EQ.1) GO TO 780
      CALL DIFFUN (T,Y,SAVE(N2,1))
      R = H/HOLD
      DO 730 I = 1,N
         Y(1,I) = SAVE(1,I)
         N11 = N1 + I
         SAVE(2,I) = HCLD*SAVE(N11,1)
730   Y(2,I) = SAVE(2,I)*R
      NG = 1
      KFLAG = 1
      GO TO 170
740   KFLAG = -1
      HNEW = H
      JSTART = NG
      RETURN

```

C*****

C* THIS SECTION SCALES ALL VARIABLES CONNECTED WITH H AND RETURNS
C* TO THE ENTERING SECTION.

C*****

```

750   RACUM = DMAX1(DAES(HMIN/HOLD),RACUM)
      RACUM = DMIN1(RACUM,CAES(HMAX/HCLD))
      R1 = 1.0
      DO 760 J = 2,K
         R1 = R1*RACUM

```

```
      DO 760 I = 1,N
760      Y(J,I) = SAVE(J,I)*R1
      F = HCLD*RACUM
      DO 770 I = 1,N
770      Y(1,I) = SAVE(1,I)
      IDCUE = K
      GC TO ( 130 , 250 , 640 ),IRET1
780      KFLAG = -4
      GC TO 470
      END
```

REFERENCES

- [1] BRAYTON, R. K., GUSTAVSON, F. G., and HACHTEL, G. D. A new efficient algorithm for solving differential-algebraic systems using implicit backward differentiation formulas. *Proc. IEEE* 60 (1972), 98-108.
- [2] BROWN, R. L. Multi-derivative numerical methods for the solution of stiff ordinary differential equations. UIUCDCS-R-74-672, U. of Illinois, Urbana, IL, August 1974. (Also Ph.D. Th., Dep. of Computer Science, U. of Illinois, 1974.)
- [3] BYRNE, G. D., and HINDMARSH, A. C. A polyalgorithm for the numerical solution of ordinary differential equations. *ACM Trans. Math. Software* 1, 1 (March 1975), 71-96.
- [4] ENRIGHT, W. H. Studies in the numerical solution of stiff differential equations. Tech. Rep. 46, Dep. of Computer Science, U. of Toronto, Toronto, Ont., Canada, Oct. 1972. (Also Ph.D. Th., Dep. of Computer Science, U. of Toronto, 1972.)
- [5] ENRIGHT, W. H. Second derivative multistep methods for stiff ordinary differential equations. *SIAM J. Numer. Anal.* 11, 2 (April 1974), 321-331.
- [6] ENRIGHT, W. H. Optimal second derivative methods for stiff systems. In *Stiff Differential Systems*, R. A. Willoughby, Ed., Plenum Press, New York, 1974, 95-111.
- [7] ENRIGHT, W. H., HULL, T. E., and LINDBERG, B. Comparing numerical methods for stiff systems of o.d.e.'s. *BIT* 15, 1 (1975), 10-48.
- [8] FORSYTHE, G. E., and MOLER, C. B. *Computer Solution of Linear Algebraic Systems*. Prentice-Hall, Englewood Cliffs, N.J., 1967.
- [9] GEAR, C. W. Algorithm 407: DIFSUB for solution of ordinary differential equations, *Comm. ACM* 14, 3 (March 1971), 176-179.
- [10] GEAR, C. W. *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, Englewood Cliffs, N.J., 1971.
- [11] HENRICI, P. *Discrete Variable Methods in Ordinary Differential Equations*. Wiley, New York, 1962.
- [12] HINDMARSH, A. C. GEAR: Ordinary differential equation solver. UCID-3001, Rev. 3, Lawrence Livermore Laboratory, U. of California, Livermore, 1974.
- [13] KNUTH, D. E. *The Art of Computer Programming*, Vol. 1: Fundamental Algorithms. Addison-Wesley, Reading, Mass., 1968.

- [14] KROGH, F. T. On testing a subroutine for the numerical integration of ordinary differential equations. *J. ACM* 20, 4 (October 1973), 545-562.
- [15] LAMBERT, J. D. Linear multistep methods with mildly varying coefficients. *Math. Comp.* 24, (1970), p. 81-94.
- [16] LAMBERT, J. D., and SIGURDSSON, S. T. Multistep methods with variable matrix coefficients. *SIAM J. Numer. Anal.* 9, 4 (December 1972), 715-733.
- [17] SHAMPINE, L. F., and GORDON, M. K. *Computer Solution of Ordinary Differential Equations: Initial Value Problems*. Freeman, San Francisco, Calif., 1975.
- [18] SHAMPINE, L. F., and GORDON, M. K. Typical problems for stiff differential equations. *ACM SIGNUM Newsletter* 10, 2 & 3 (November, 1975), 41.
- [19] SKEEL, R. D. Equivalent forms of multistep formulas. In preparation, Dep. of Computer Science Tech. Rep., U. of Illinois, Urbana, IL.

U. S. ATOMIC ENERGY COMMISSION
UNIVERSITY-TYPE CONTRACTOR'S RECOMMENDATION FOR
DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

1. AEC REPORT NO. C00-2383-0030	2. TITLE BLENDED LINEAR MULTISTEP METHODS
--	--

3. TYPE OF DOCUMENT (Check one):

- ☒ a. Scientific and technical report
- ☐ b. Conference paper not to be published in a journal:
Title of conference _____
Date of conference _____
Exact location of conference _____
Sponsoring organization _____
- ☐ c. Other (Specify) _____

4. RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

- ☒ a. AEC's normal announcement and distribution procedures may be followed.
- ☐ b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.
- ☐ c. Make no announcement or distribution.

5. REASON FOR RECOMMENDED RESTRICTIONS:

6. SUBMITTED BY: NAME AND POSITION (Please print or type)

C. W. Gear
Professor and Principal Investigator

Organization

University of Illinois at Urbana-Champaign
Department of Computer Science
Urbana, IL 61801

Signature

Charles W. Gear

Date

April 1976

FOR AEC USE ONLY

7. AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

8. PATENT CLEARANCE:

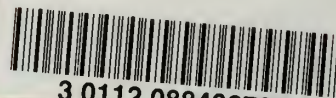
- ☐ a. AEC patent clearance has been granted by responsible AEC patent group.
- ☐ b. Report has been sent to responsible AEC patent group for clearance.
- ☐ c. Patent clearance not required.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-76-800	2.	3. Recipient's Accession No.
	Title and Subtitle BLENDED LINEAR MULTISTEP METHODS		5. Report Date May 1976
Author(s) Robert D. Skeel and Antony K. Kong		6.	
Performing Organization Name and Address Department of Computer Science University of Illinois at Urbana-Champaign Urbana, IL 61801		8. Performing Organization Repr. No. UIUCDCS-R-76-800	
2. Sponsoring Organization Name and Address US ERDA Chicago Operations Office 9800 South Cass Avenue Argonne, IL 60439		10. Project/Task/Work Unit No.	
		11. Contract/Grant No. US ERDA AT(11-1) 2383	
3. Supplementary Notes		13. Type of Report & Period Covered	
4. Abstracts The accuracy of linear multistep formulas suitable for stiff differential systems is limited. Greater accuracy can be attained by including higher derivatives in the formula, but this is not practical for all problems. However it is possible to duplicate the absolute stability region for any given m-derivative multistep formula by taking a linear combination of m-multistep formulas. These blended formulas are similar to Lambert and Sigurdsson's linear multistep formulas with variable matrix coefficients, but the approach is different. Implementation details and numerical results are presented for a blend of the Adams-Moulton and the backward differentiation formulas.		14.	
5. Key Words and Document Analysis. 17a. Descriptors ordinary differential equations stiff equations multistep methods Adams method absolute stability linear multistep methods			
b. Identifiers/Open-Ended Terms			
c. COSATI Field/Group			
6. Availability Statement Unlimited		19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 70
		20. Security Class (This Page) UNCLASSIFIED	22. Price

JUN 21 REC'D



UNIVERSITY OF ILLINOIS-URBANA
510.84 IL6R no. C002 no. 800-805(1976)
Internal report /



3 0112 088402786